Estimate a probability density function
ML_2022: Machine Learning

https://people.sc.fsu.edu/~jburkardt/classes/ml_2022/probability_lab/probability_lab.pdf

Two flies land at random points in the unit circle. On average, what is the distance between them?

Estimate a Probability Density Function!

We will estimate the answer to a probability question the experimental way. We will simply look at many random examples of the fly in the circle problem, and average the results. This exercise will give us practice in using Python to use random numbers, create user functions, work with arrays and iteration, and create an interesting plot.

- *Random numbers can be used to study an event with many possible outcomes;*
- *A histogram summarizes the number of times each outcome occurs;*
- *When the outcomes can be a range of real numbers, we have a probability density function;*
- *The shape of our histogram should approximate the exact PDF;*
- *Plotting will be a big help in catching mistakes and understanding results;*

1 Plot a circle and some random points

Our stated problem involves two flies landing at random points in a unit circle. We'd like to use Python to draw a picture of this situation, so we need to know:

- how to convert the mathematical circle into a "computational" circle;
- how to get access to the graphics library;
- how to draw a sequence of points as a connected curve;
- how to generate random points;
- how to add an image of those points to our plot;
- how to save a copy of our work as a `jpg` file;

Although we may know the formula for the points (x, y) that lie on the unit circle, to make a plot we actually need to be able to generate a sequence of such points, and connect them. Luckily, as the angle θ goes from 0 to 2π, the coordinates of the points are $(\cos(\theta), \sin(\theta))$. We want to pick n equally spaced values of θ and create a list of the corresponding points.

We could use a `for` loop to do this:

```
import numpy as np

cn = 100
cx = np.zeros ( cn )
cy = np.zeros ( cn )
```

```
for i in range ( 0, cn ):
  theta = 2.0 * np.pi * i / cn
  cx[i] = np.cos ( theta )
  cy[i] = np.sin ( theta )
```

but a cleaner alternative is available, which sets up each array with a single command:

```
import numpy as np

cn = 100
theta = np.linspace ( 0, 2.0 * np.pi, cn )
cx = np.cos ( theta )
cy = np.sin ( theta )
```

To do any kind of plotting we need to import `matplotlib`. Here is a basic plot of the circle:

```
import matplotlib.pyplot as plt
plt.plot ( cx, cy )
plt.show ( )
```

and here is a fancier version which corrects the aspect ratio, chooses a color for the lines, and saves a copy of the plot.

```
import matplotlib.pyplot as plt
plt.plot ( cx, cy, 'b-', linewidth = 3 )
plt.grid ( True )
plt.title ( 'A circle' )
plt.show ( )
```

You may notice that your circle is not circular. That's because the matplotlib drawing space is a sort of rectangle, which has the effect of stretching things in the horizontal direction. To cancel this effect, try

```
plt.axis ( 'equal' )
plt.show ( )
```

We want to create two flies at random positions. The `numpy` function `rand()` returns a random value between 0 and 1, and this is almost what we want. In fact, let's create 20 random flies, to get a feeling for how this works.

```
fx = rand ( 20 )
fy = rand ( 20 )
plt.plot ( fx, fy, 'r*', markersize = 5 )
plt.show ( )
```

Do you see that our random flies aren't well positioned? They are all in the upper right quadrant of the circle; that's because, as we said, `rand()` always returns values between 0 and 1. We can try again by doubling the random values and subtracting 1. We will have to use a different color for our flies, because they will show up in the same plot.

```
fx = 2 * rand ( 20 ) - 1
fy = 2 * rand ( 20 ) - 1
plt.plot ( fx, fy, 'g*', markersize = 5 )
plt.show ( )
```

Now we want to change our title, and save a copy of our plot:

```
plt.title ( 'Two random flies in a circle' )
filename = 'fly_plot.jpg'
plt.savefig ( filename )
plt.show ( )
plt.close ( )
```

You may have typed these commands interactively, but now it's time to create a single file called *fly_plot.py*, and store the useful commands in this script:

```
import numpy as np

cn = 100
theta = np.linspace ( 0, 2.0 * np.pi, cn )
cx = np.cos ( theta )
cy = np.sin ( theta )

import matplotlib.pyplot as plt
plt.plot ( cx, cy, 'b-', linewidth = 3 )
plt.grid ( True )
plt.axis ( 'equal' )

fx = 2 * np.random.rand ( 2 ) - 1
fy = 2 * np.random.rand ( 2 ) - 1
plt.plot ( fx, fy, 'g*', markersize = 5 )

plt.title ( 'Two random flies in a circle' )
plt.savefig ( 'fly_plot.jpg' )
plt.show ( )
plt.close ( )
```

Notice that we have used the `plt.plot()` command twice, but the second item (the flies) was simply plotted "on top of" the first item (the circle). If you actually wanted a new plot, you could issue one of the commands

- `plt.clf()`, which blanks out the current plot;
- `plt.close()`, which discards the current plot and closes the plotting window;
- `plt.figure()`, which opens up a separate figure for new plot commands.

2 Exercise 1:

Create the file `fly_plot.py`, run it, and check that the plot `fly_plot.jpg` looks reasonable.

3 Uniform Random Points in a Circle

To simulate the problem, we are going to we need to select points from the circle in a uniform random way. That means that the probability of drawing from any particular region should be proportional to its area. If we generate 100 points in the circle, they should seem to be sprinkled more or less evenly over the surface.

To estimate the typical distance between the flies, we are going to simulate the situation many times, each time placing the two flies randomly in the circle. In order to make such random choices, the `numpy` module offers the function `numpy.random.rand()`, which is used as follows

```
import numpy as np
x = np.random.rand           # Big mistake!  Don't do this!  x is now a new name for rand.
y = np.random.rand ( )       # A random value.
z = np.random.rand ( 1 )     # A random value.
v = np.random.rand ( 5 )     # A random (row) vector of 5 elements
A = np.random.rand ( 4, 3 )  # A random array of 4 rows and 3 columns
```

This random number generator produces a real values between 0 and 1. How can be use it to generate points in the circle? Here are three proposals. In each case, we generate two random values, r_1 and r_2, and then:

1. $r = r_1, \theta = 2\pi r_2, x = r\cos(\theta), y = r\sin(\theta)$
2. $r = \sqrt{r_1}, \theta = 2\pi r_2, x = r\cos(\theta), y = r\sin(\theta)$

3. $x = 2 * r_1 - 1, y = 2 * r_2 - 1$, but try again if $1 < x^2 + y^2$.

The functions cos(), sin(), and $\sqrt{}()$ are available in `numpy`. Assuming we have already issued the statement `import numpy as np`, we can use these functions in formulas as long as we preface their names with `np.`, as in `x = r * np.cos(theta)`. If a variables `r` and `theta` are arrays, then this command will automatically create a corresponding array `x`.

An array is simply a list of values. We are going to generate arrays `x,y` of length `n`. In some cases, we can give a command that operates on the entire array; in other cases (such as method #3 above) we may have to look at a particular entry in an array. If an array `x` has `n` values, then the first, `i`-th, and last entries are `x[0]`, `x[i]`, and `x[n-1]`. To set all entries of `x` to random values, the following are equivalent:

```
x = np.random.rand ( n )
```

and

```
x = np.zeros(n)
for i in range ( 0, n ):
  x[i] = np.random.rand ( )
```

For methods #1 and #2, we can avoid using a `for()` loop, but for method #3, it will be necessary to check each entry of the arrays. The easiest way to do this is to start by initializing `x` and `y` using vector statements:

```
x = 2 * np.random.rand ( n ) - 1
y = 2 * np.random.rand ( n ) - 1
```

Then you can check that each pair (x[i],y[i]) is actually inside the unit circle and if not, try again, something like this:

```
for i in range
  while ( ??? x[i], y[i] not in unit circle...):
    x[i] = ???  (get one new random value in [-1,+1])
    y[i] = ???  (same)
```

The `while()` statement:

- checks the condition;
- if the condition is true, it carries out the following (indented) statements;
- goes back to the while statement and checks again (and again and again if necessary);

Now you should know enough to be able to put together Python commands that will sample `n` points from the unit circle, and plot them. It would be nice if, in addition to the sample points, you include a plot of the unit circle as well!

4 Exercise 2:

Generate and plot `n`=1000 points in the unit circle:

1. for method #1;
2. for method #2;
3. for method #3;

Judging from your plots, which methods seem to sample the circle uniformly?

5 Average Distance in the Circle

We wish to do a simulation that computes an estimate `dmean` for the average distance between a pair of uniformly random points in the unit circle. If we have a vector of `n` distances, the average can be defined by

$$\text{dmean} = \frac{1}{n} \sum_{i=0}^{n-1} d_i$$

We also want an estimate `dvar` for the variance, σ^2, defined by

$$\text{dvar} = \frac{1}{n-1} \sum_{i=0}^{n-1} (d_i - \text{dmean})^2$$

We already have practiced most of the things we need in order to carry out this task. Here's an outline of how to do it:

1. Choose a value `n`;
2. Use method #2 or #3 to compute arrays `x1`, `y1`, `x2`, `y2`, each of length `n`;
3. Use `np.zeros()` to set up an array `d`, of length `n`, for the distances;
4. Using a `for()` loop, compute `d[i]` as the square root of the distance between `(x1[i],y1[i])` and `(x2[i],y2[i])`;
5. Use the functions `np.mean()` and `np.var()` to compute `dmean` and `dvar`;

6 Exercise 3:

Write a program that:

- Generates `n` random pairs of points in the unit circle;
- Determines the `n` distances `d` between pairs of points.
- Computes and prints `dmean = mean ( d );`
- Computes and prints `dvar = var ( d );`

Theoretically, the mean value should be dmean $= \frac{128}{45\pi}$ and the variance should be dvar $= \frac{2025\pi^2 - 128^2}{45^2\pi^2}$. If your results are not close to these values, check your program. If your program seems correct, what can you do to try to get a better approximation?

7 The PDF for distances

Let us write $p(d)$ to represent the probability of observing the distance d between a pair of randomly chosen points in the unit circle. We know then that $p(d)$ is nonzero only for $0 \leq d \leq 2$, and that $\int_0^2 p(d) = 1$, and we have seen that the average value of d is near 1. To get a better feeling for the variation of d, we can generate a large number n of cases, compute d for each one, and then construct a histogram.

Usually, a histogram simply divides the range into a number of intervals, and displays a count of the number of times d falls into each interval. However, if we divide this count by n, the number of cases, the resulting normalized histogram will approximate the probability density function for d, that is, the function $p(d)$.

We have already done almost all the work necessary to make this plot. The only remaining issue is how to make a histogram. Luckily, the module `matplotlib` has a function that will do this for us, with a couple interesting arguments:

```
plt.hist ( d, bins = ?, density = True )
```

Here `bins` is the number of subintervals into which the range will be subdivided. Use your judgment to choose an appropriate value. The argument `density = True` will do the normalization discussed above, so that we match the shape of the PDF.

8 Exercise 4:

- Use circle sampling method #2 to generate `n` pairs of points;
- Compute the distance vector `d`.
- Use the **hist()** command to create a histogram of the data, choosing the number of bins, and use the `density = True` normalization;
- Use the `linspace()` function to define a vector `d2` that samples the range from 0.0 to 2.0.
- Evaluate the exact PDF on the points `d2` by

$$p_2 = \frac{1}{\pi} d_2 \left(4 \arccos(d_2/2) - d_2 \sqrt{4 - d_2^2}\right)$$

  (You will need the function `np.arccos()` here.)
- Add the PDF curve `(d2,p2)` to your histogram using the `plot()` command.

If you chose `n` large enough, and have a reasonable number of bins in your histogram, your histogram and the PDF curve should come close to matching.