Computer Programming for Engineering

Python Project Report 2

Name: Khadijatu Nayi Alhassan

Student ID: 73692022

Date due: April 28, 2019.

Contents

Introduction:

The purpose of this project was to write a python program that implements the four popular probability distribution concepts in probability theory and statistics. These concepts are the Normal distribution (Gaussian distribution), the Poisson distribution, the Geometric distribution, and the Exponential distribution. The motivation for this project was to combine the concepts I learned from my A-level statistics class and Computer programming for Engineering class. I believe using technology to solve such problems will help students who have an interest in studying probability and statistics.

Application of knowledge:

I used Object Oriented Programming (OOP) to make my code modular and handy. The whole project is defined in a class named **Probability**. This class has five instance methods namely:

1. normal_dist1: a method for calculating the probability of continuous variables given the mean, the standard deviation and the x value, where x follows a normal distribution.
2. Normal_dist2: a this method for calculating the probability given a range of values (X1 and X2), the mean and the standard deviation.
3. geometric: a method that stimulates the geometric distribution. The geometric distribution represents the number of failures you get before obtaining a success in a series of trials.
4. exponential: this method stimulates the exponential distribution.
5. poisson: this method stimulates the Poisson distribution.

The constructor has no parameters hence to create an instance of this class, you will do something like: *instance_one = Probability ()*. The variable *instance_one* is called an object or instance of the class Probability. However, the class has three instance variables namely: **self. constant, self. f_x** and **self.x.** Where *self. constant* is the instance variable for the constant of the normal distrubtion formula (shown in fig3 in the appendix) , *self.f_x* is the normal distribution formula (show in fig3 in the appendix) and *self.x* is a sympy symbol. The product of **self. f_x** and self. constant gives the Probability Density Function of the normal distribution.

normal_dist1 and normal_dist2

The *normal_dist1* method takes a mean of a distribution, the standard deviation and x1. The mean and standard deviation are characteristics of the normal distribution and x1 is a value in the data set. normal_dist1 calculates the probability for a one-tail case. The limit of integration apart from infinity is the z-value which is calculated as z = (x1 – mean)/standard deviation. One tail case is shown in the appendix.

The **normal_dist2** also works just like **normal_dist1**. The only difference is that it is used for calculating the probability in a two-tail case, thus, it has two limits of integration. It takes the mean, the standard deviation, x1 and x2 as parameters. To find the lower and the upper limits of the

integration in **normal_dist2**, I used list methods *max and min* which is under chapter 4, page 129 of the Python book. Two tail case is shown in the appendix.

Plot_norm_dist

The plot_norm_dist method creates a plot of the normal distribution curve and shades the region under the curve based on the parameters. It takes the mean and standard deviation to calculate the z-value. The z-value(s) act as the limit to which the area under the curve is shaded to. The shade_lt creates a plot to shade all values less than or equal to the z-value. The shade_gt creates a plot to shade all values greater than or equal to a z-value. The shade_bw shades the area between two z-values. This is how the plot_norm_dist works: NumPy linspace is used to create a range of all x-values. A for loop is used to substitute each x-value into the normal_dist1 function. The function returns two values; probability less than the x-value and probability greater than the x-value. For each x-value, probability less than the value is appended to y_1 and probability greater than the value is appended to y_2. At the end of the loop, y_1 and y_2 are merged to form the list of all y-values. x_1 is plotted against y_1 to give the full curve. Shade_lt, shade_gt and shade_bw all source values from x_1 and y_1 based on some conditions.

geometric

The next method is geometric, and it takes p, a probability of success and x1, a value in the data set as parameters. It returns the probability based on calculation with the geometric distribution equation $p(X>x) = (1-p)^{x1}$. This equation changes depending on the probability, whether greater than, less than, greater than or equal to, etc. It also calculates and returns the expectation and variance of the geometric distribution.

exponential

The probability of an exponential distribution is calculated using the formula $P(X < x) = \mathbf{1} - e^{-\lambda x}$. The exponential method takes L, x1 and sometimes x2 as the parameters. L corresponds to the Greek symbol lambda, which is the mean of the exponential distribution. x1 and x2 are values in the data set of the exponential distribution. The user chooses if he wants the probability calculated with one value or two values, so I made x2 a keyword argument. In the case where the user wants the probability with just one value, x2 is assigned None. Here I used selection statements which is under chapter 2, page 67. Only probability of less than and greater than this value is calculated, with expected value and variance of the exponential distribution. If the user wants to input two values, x2 is assigned a value and the probability between two limits is calculated.

Poisson

The Poisson distribution uses the formula $P(x; \mu) = (e^{-\mu})(\mu^{x}) / x!$ The method poisson takes L which is the mean and x1, a value in the data set of the poisson distribution. In the method, I used for loops, which is under chapter 1, page 33 and accumulator pattern under chapter 2, page 54.

The calculations in each method is returned in a dictionary, applied from chapter 4, page 135.

Third Party Libraries

Apart from the Python in-built math module, I installed other libraries including *sympy, numpy and matplotlib*. I installed these libraries by typing *pip install (name of the library)* in the command prompt.

Appendix

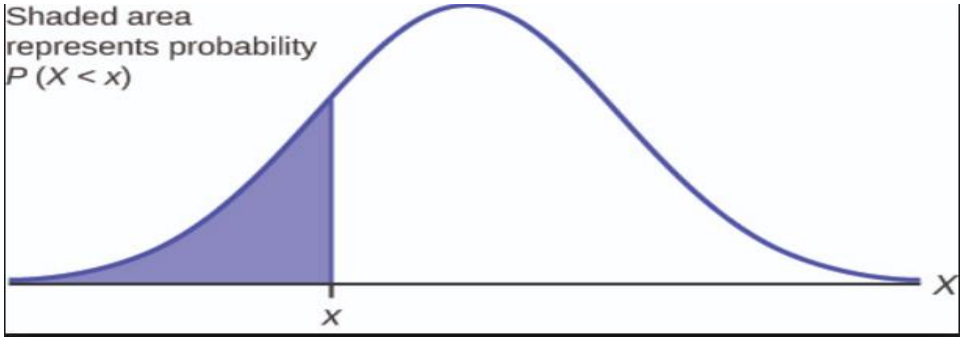

Fig 1: One tail case.

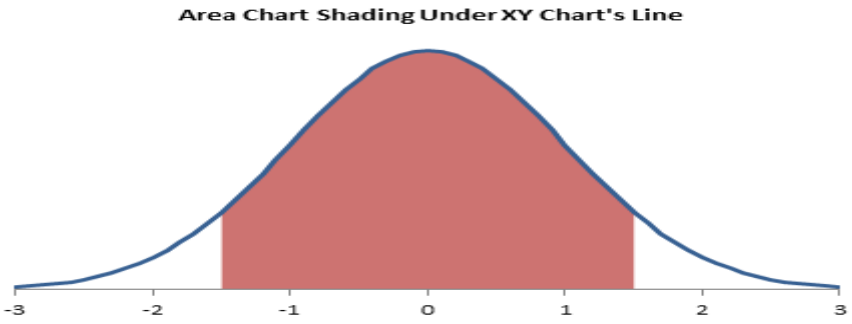

Fig 2

Normal Probability Density Function

$$F(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2 / 2\sigma^2}$$

Fig 3

References

Fig.1

One tail normal distribution. Retrieved from

https://www.google.com/search?rlz=1C1EJFC_enGH817GH817&tbm=isch&q=one+tail+normal+distribution&chips=q:one+tail+normal+distribution,online_chips:standard&usg=AI4_-kRqxTw3XirCj2mnT8l5cC2Ve-EOPw&sa=X&ved=0ahUKEwixx9KByKzhAhVIz4UKHcX1D88Q4lYIJygB&biw=1486&bih=631&dpr=1.25#imgdii=F878nAumj4lufM:&imgrc=lKv3-pxVNaLLpM:

Fig.2

Normal distribution between two values. Retrieved from

https://www.google.com/search?rlz=1C1EJFC_enGH817GH817&tbm=isch&sa=1&ei=28ugXMrIHsGalwTz4qeICA&q=normal+distribution+between+two+values&oq=normal+distribution+betwee&gs_l=img.1.0.0j0i24l9.34655.41153..45058...0.0..1.316.2858.0j3j6j3.....1....1..gws-wiz-img.....0i7i30j0i8i7i30j35i39j0i67j0i8i30.vAOUvlRsRpk#imgdii=fEOCnsJmhNm5YM:&imgrc=axiQ3kigzj58WM:

Fig.3

Normal Distribution formula. Retrieved from

https://www.google.com/search?q=normal+distribution+formula&safe=active&client=firefox-b-d&tbm=isch&source=iu&ictx=1&fir=fXPMiA1IFZmoKM%253A%252CwVxnWr67MFgp5M%252C_&vet=1&usg=AI4_-kT1Z9ddaQM1BEHqxFsEkSM1GQJs3g&sa=X&ved=2ahUKEwiCjru1pa3hAhWtY98KHfruAKUQ9QEwAHoECAsQBg#imgrc=fXPMiA1IFZmoKM: