Lab 4 - Continuous Random Variables and Confidence Intervals

March 19, 2020

1 Preamble

As in previous Labs, you will find your personalised Lab exercises on Brightspace, under Assessment → Quizzes → Lab 4. Read this document and ensure that you understand this week's materials before attempting your Quiz. All of the functions you need for answering your Quiz questions are given in this document, or were covered in previous labs.

In this week's lab we will need the following packages. Make sure, to import in before you start.

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.stats as stats
```

The deadline for submitting your Quiz attempt and Python file is **Monday March 30th at 12pm**.

2 Continuous Random Variables

2.1 Normal Distribution

The normal distribution is the most important distribution in statistics. The probability density of the normal distribution is of the form:

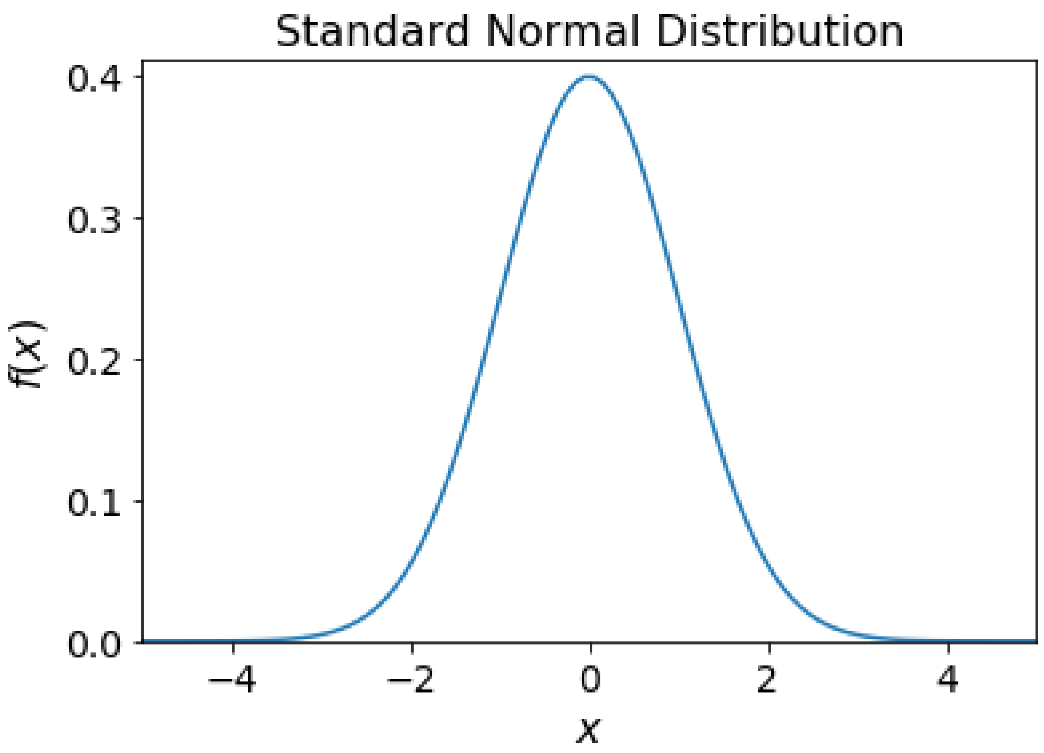
$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right]$$

The code below create a graph of the density curve when $\mu = 0$ and $\sigma = 1$. How would you change the code to plot the density curve for $\mu = 5$ and $\sigma = 10$?

Note: The *scale* argument is the standard deviation σ. We usually characterise a Normal distribution as $X\ N(\mu, \sigma^2)$ but `norm` methods take the standard deviation as an input, not the variance.

[2]:
```
x = np.linspace(-5,5,100)
plt.figure()
plt.plot(x,stats.norm.pdf(x,loc=0,scale=1))
plt.axis([-5,5,0,0.41])
plt.xlabel('$x$',fontsize=16)
plt.ylabel('$f(x)$',fontsize=16)
plt.xticks(fontsize=14)
plt.yticks([0,0.1,0.2,0.3,0.4],fontsize=14)
plt.title('Standard Normal Distribution',fontsize=16)
```

[2]:
```
Text(0.5, 1.0, 'Standard Normal Distribution')
```



As in the previous lab, we use the **rvs** method to select a random sample from this distribution. The code below simulates 1000 values from the standard normal distribution $N(0,1)$, i.e. creates a sample of size 100.

[ ]:
```
sample = stats.norm.rvs(loc=0,scale=1,size=10)
```

Exercise 1: Change the above code to create a sample of size 1000 from a normal distribution with $\mu = 10$ and $\sigma = 0.5$. Plot a histogram of these values and note the shape. Recall that for normally distributed data the histogram should be bell shaped and symmetric.

How does the shape of the histogram change if you increase/decrease: 1. the mean μ? 2. the standard deviation σ? 3. the sample size n?

Note: You can adjust the number of bins used in plotting a histogram in Python by including the argument bins. For example if you want 30 bins in your histogram run:

[ ]:
```
plt.hist(sample,bins=30)
```

You can also use a list/array as the bins argument. This will use the list elements as start and end point for your bins. For example, the code below creates a histogram with 6 bins; -3 to -2, -2 to

-1, -1 to 0, 0 to 1, 1 to 2 and 2 to 3. When comparing to distributions it is good practice to use the same bins for both.

[ ]:
```
plt.hist(sample,bins=[-3,-2,-1,0,1,2,3])
```

As we learnt in lectures, it is not possible to work out probabilities for the normal distribution by integrating the probability density function. Instead, we used the *New Cambridge Statistical Tables* to compute probabilities. We can also use statistical software, such as Python, to compute these probabilities. The code below computes the probability $P(X < 0.5)$, where $X \sim N(0,1)$. Does this value match the value in your tables?

[ ]:
```
prob1 = stats.norm.cdf(0.5,loc=0,scale=1)
print(prob1)
```

Python will calculate probabilities with negative numbers. When using the *New Cambridge Statistical Tables* we change the signs and direction of the inequality ($P(X < -1) = P(X > 1)$), you do not need to do that here. For example, the code below calculates $P(X < -1)$, where $X \sim N(0,1)$.

[ ]:
```
prob2 = stats.norm.cdf(-1,loc=0,scale=1)
print(prob2)
```

Exercise 2: How would you calculate the following probabilities using Python? 1. $P(X < 1.95)$ where $X \sim N(0,1)$ 2. $P(X > 2.5)$ where $X \sim N(0,1)$ 3. $P(X < -0.8)$ where $X \sim N(0,1)$ 4. $P(1.2 < X < 4)$ where $X \sim N(0,1)$

Note `cdf` gives the probability of being less than the specified value. The probability of being greater than a particular number is 1 minus the probability of being less than that number $P(X > a) = 1 - P(X < a)$. Alternatively, you could note that $P(X > a) = P(X < -a)$ and compute $P(X < -a)$.

Now suppose we want to compute $P(X < 11)$ for a non-standard normal random variable $X \sim N(10,9)$. If we were using the *New Cambridge Statistical Tables*, we would have to standardise before answering this question. But with Python we don't need to do that! The `norm.cdf` allows you to specify μ (*loc*) and σ (*scale*).

[ ]:
```
prob3 = stats.norm.cdf(11,loc=10,scale=3)
print(prob3)
```

Remember that the varaince is given in $N(\mu,\sigma^2)$, while the *scale* argument in `norm.cdf` is the standard deviation σ. Hence, the scale is 3 in the above code ($\sqrt{9}$).

Exercise 3: Write a piece of Python code to compute the following probabilities: 1. $P(X < 10)$ where $X \sim N(8,16)$ 2. $P(X > 0)$ where $X \sim N(-7,4)$ 3. $P(1 < X < 2.5)$ where $X \sim N(2,0.25)$

Note: If the mean is 4 and you want to know $P(X < 3)$ you should expect the probability to be less than 0.5, because 3 is less than 4 and $P(X < 4) = 0.5$. Hence, the area to the left of 3 is smaller than the area to the left of 4, $P(X < 3) < P(X < 4)$. Conversely, if you want to know $P(X > 3)$, you should get an anwser greater than 0.5, because the area to the right of 3 is larger than the area to the right of 4, $P(X > 3) > P(X > 4)$. You can make a similar argument for numbers greater than the mean, e.g. $P(X > 6)$ and $P(X < 6)$. Draw a pictures to ensure you are calculating the correct probabilities and obtaining a sensible answer.

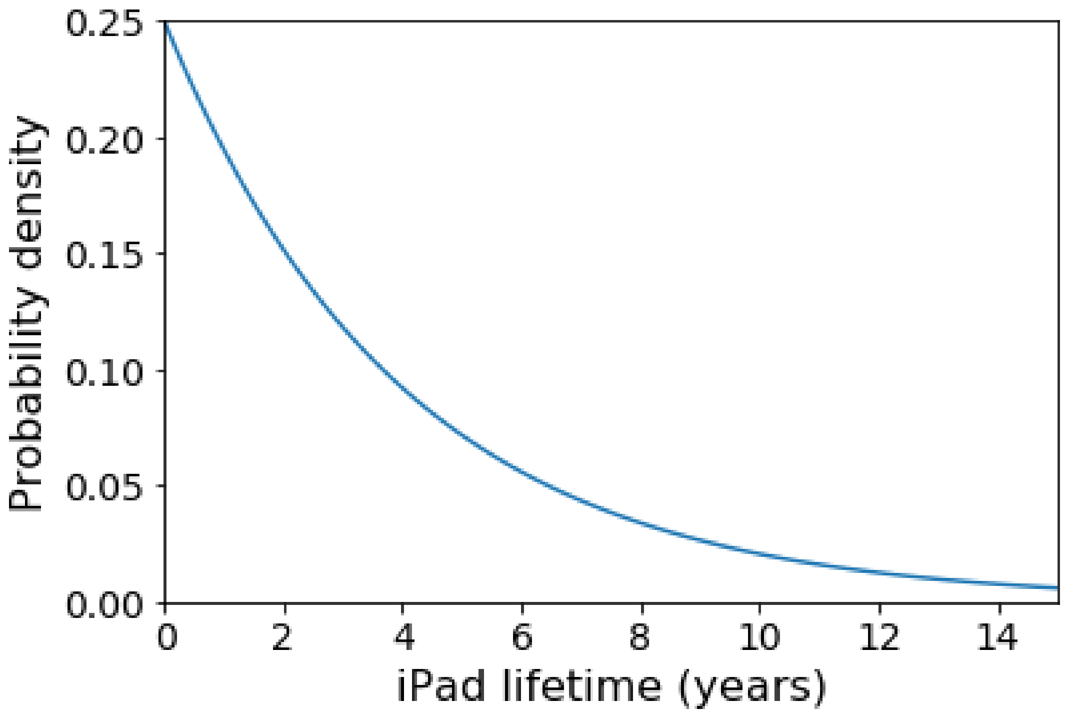
Exercise 4: Assume that the lengths of baby elephants' trunks follow a normal distribution with mean 1.2 metres and variance 0.1 metres. What is the probability that a baby elephant will have a trunk between 1.4 and 1.8 metres long?

2.2 Exponential Distribution

Suppose that the lifetime of an iPad (let's call this random variable *iTime*) is well modelled by an exponential distribution with a mean lifetime of 4 years. Thus $iTime \sim \exp(\lambda = 0.25)$.

[3]:
```
x = np.linspace(0,15,100)
plt.figure()
plt.plot(x,stats.expon.pdf(x,scale=4))
plt.axis([0,15,0,0.25])
plt.xlabel('iPad lifetime (years)',fontsize=16)
plt.ylabel('Probability density',fontsize=16)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
```

[3]:
```
(array([0.  , 0.05, 0.1 , 0.15, 0.2 , 0.25]),
 <a list of 6 Text yticklabel objects>)
```



Note: The scale parameter for `expon` functions in Python is $1/\lambda$. Hence, in the example *scale* is 4.

As with the normal distribution, can simulate some sample lifetimes from this distribution by applying the `rvs` method to the distribution. The code below takes a sample of size 50 from an exponential distribution with $\lambda = 0.25$, i.e. a sample of 50 iPads with a mean lifetime of 4 years.

```
[ ]: samp_exp = stats.expon.rvs(scale=4,size=50)
```

Exercise 5: 1. Plot a histogram of this sample data. Does the histogram resemble the curve above? 2. Take a new sample and remake the histogram. Does it look the same? 3. Increase the sample size and create a histogram of the sample data. How does the plot change?

We can calculate probabilties from an exponential distribution by applying the `cdf` method to the distribution. Suppose we are interested in knowing what the probability is that an iPad will function for less than 5 years. We can quickly calculate this using the following command:

```
[ ]: prob_exp1 = stats.expon.cdf(5,scale=4)
     print(prob_exp1)
```

Exercise 6: Now suppose that *iTime* is actually exponentially distributed with a mean of 5 years. What is the probability that an iPad is functional for between 6 and 8 years?

3 Confidence Intervals

In lectures we learnt how to calculate confidence intervals for the population mean μ given a large sample. The $100(1-\alpha)\%$ confidence interval for μ is given by:

$$\bar{x} \pm Z_{\frac{\alpha}{2}} \frac{s}{\sqrt{n}}$$

where $\bar{x}$ is the sample mean, s is the sample standard deviation and n is the sample size. Remember $Z_{\frac{\alpha}{2}}$ is the value of the standard normal random variable Z such that $\mathbb{P}(Z > Z_{\frac{\alpha}{2}}) = \frac{\alpha}{2}$. This value can be calculated with the `norm.ppf()` function, from `stats`.

Now if we are given $\bar{x}$, s and n we can compute the confidence interval as follows:

```
[ ]: xbar = 1.37
     s = 0.10
     n = 50
     confidence = 0.95
     z = stats.norm.ppf((1 + confidence)/2)

     lower = xbar - z*s/np.sqrt(n)
     upper = xbar + z*s/np.sqrt(n)

     print([round(lower,3),round(upper,3)])
```

Exercise 7: Change the above code to compute the 95% confidence interval for a sample with $\bar{x} = 100$, $s = 15$ and $n = 50$. Now compute the 99% confidence interval instead. Comment on the width of the two intervals.

Download the dataset called `variables.csv` from Brightspace, and save it to the folder that your Jupyter Notebook is saved in. Now import it into your notebook using the `read_csv()` function from Pandas.

```
[ ]: data = pd.read_csv('variables.csv')
     print(data.head())
```

We can calculate the mean and standard deviation, using `np.mean()` and `np.std()`, respectively. The sample size is the length of the data. The dataset has multiple columns, each corresponding to a different variable. The code below finds the mean and standard deviation for the first column and hence, the confidence interval for the first column.

```
[ ]: xbar = np.mean(data.iloc[:,0])
     s = np.std(data.iloc[:,0])
     n = len(data)
     confidence = 0.95
     z = stats.norm.ppf((1 + confidence)/2)

     lower = xbar - z*s/np.sqrt(n)
     upper = xbar + z*s/np.sqrt(n)

     print('Confidence interval for the true population mean:
      ↪',[round(lower,2),round(upper,2)])
```

Exercise 7: Change the above code to find the 99% confidence interval for the second column.

Remember that indexing in Python starts at 0, not 1. Hence, the index for the second column is 1, not 2.

```
[ ]:
```