Practical Session 5, 2019/2020 (B)

1. Aim of the Session

In this session, you will learn 1) how to calculate probabilities in Python; 2) how to train a Naïve Bayes Classifier and how to use the trained model to do predictions on an unseen dataset; 3) what object-oriented programming is and how you can write your own classes and create objects from them.

2. Probability

- Uniform distribution

  Task1: Download probability.ipynb from Canvas. The aim of this program is to find the probability of an even number appearing in an integer space [2, 50]

  1) To run the program, you need to fill in **????** with correct code first. Note that P(E) = n(E)/n(S), n(E) and n(S) are the number of the set E, the event, and the number of the set S, the sample space, respectively.
  2) Check the probability

- Probability of event A or event B

  Suppose there are two possible events and you want to find the probability of either one of them happening. Recall calculating the probability of either set of outcomes, you can find the probability of the union of the two sets.

  For example, for a simple and fair die roll (the sample space is {1, 2, 3, 4, 5, 6}), you consider the following two events:

  A = The number is a prime number
  B = The number is even.

  You need to calculate the probability that you get either a prime number or an even number. The following python code shows how to generate the sample space and two events using set.

```
from sympy import FiniteSet
S = FiniteSet(1, 2, 3, 4, 5, 6)
a = FiniteSet(2, 3, 5)
b = FiniteSet(2, 4, 6)
```

  Task2:

  1) Use the *union()* method to find the event set
  2) Calculate the probability of union of the two sets in Python. The probability is defined by (the number of events)/(the number of samples in the sample space).

- Probability of event A and event B

Similar to above, you can use the *intersect()* method to calculate the chances of both of events happening. For example, the chances that a die roll is both prime and even.

Task3:

1) Calculate the probability that a 6-sided die roll is both prime and even using a python program.

- Generating uniform random numbers

  To generate a random integer, you need to import the **random** module first and then call the *randint()* function, which takes two integers as arguments and returns a random integer that falls between these two numbers (both inclusive). For example,

```
import random
random.randint(1,6)
```

Task4:

1) Generate an integer in between 5 and 10.

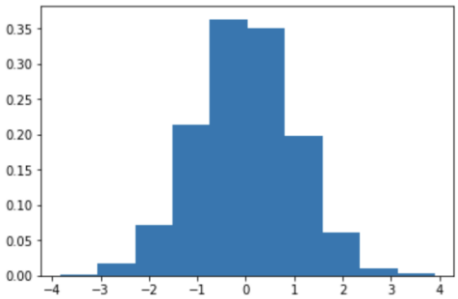
3. Probability Distributions with SciPy

To better understand probability distributions, the best way is to simulate random numbers or generate random variables from specific probability distribution and visualising them. The **scipy.stats** module provides a large number of probability distributions. More details can be viewed here https://docs.scipy.org/doc/scipy/reference/stats.html.

For example, the following code generates 5000 random numbers from a Gaussian distribution with zero mean and unit standard deviation and displays a histogram. You can use **norm.rvs?** and **ax.hist?** to help you understand how these functions work.

```
from scipy.stats import norm
import matplotlib.pyplot as plt

r = norm.rvs(size=5000)
fig, ax = plt.subplots(1,1)
ax.hist(r, density=True, histtype='stepfilled')
```



Task5:

1) Generate 5000 data points from a Gaussian distribution with mean=30 and standard deviation=10. Observe the histogram to see if you have got the correct mean value, that is if the center of the histogram is 30.
2) Generate 5000 Bernoulli random numbers with success probability p =0.2 and visualisate the data using a histogram plot. The function you can use is

bernoulli.rvs. Use **bernoulli.rvs?** to help you understand how this function works.

3) Generate 10000 data from binomial distribution and visualisate the data using a histogram plot. The function you can use is **binom.rvs**. Recall that Binomial distribution is a discrete probability distribution like Bernoulli. It can be used to obtain the number of successes from multipl (**N**) Bernoulli trials. In this task, supppose you want to find the number of successes in 20 Bernoulli trials with p =0.6 and you want to repeat 10000 times. Can you explain what you can see from the obtained histogram plot?
4) Generate 5000 random numbers from Poisson random variables with $\mu = 0.2$ using **poisson.rvs** and plot them. A Poisson distriubtion has the form like this: $P(k \text{ events in an interval}) = e^{-\lambda}\frac{\lambda^k}{k!} \quad k = 0,\ 1,\ \cdots. \lambda$ is the average number (μ) of events per interval; e is the number 2.71828. A Poisson distribution helps in describing the chances of occurrence of a number of events in some given time interval or given space conditionally that the value of average number of occurrence of the event is known.

4. Conditional Probability

The conditional probability is defined as follows:

$$P(B \mid A) = \frac{P(A \text{ and } B)}{P(A)} \qquad P(A) > 0$$

In addition, $P(A \cup B)$ is given by:
$P(A \cup B) = P(A) + P(B) - P(A \cap B) = P(A) + P(B)$

In a class of 100 students, 10 of them failed mathematics, 15 of them failed physics, and 8 of the students failed both mathematics and physics. A student is selected at random.

Task6: Let A ={students failed mathematics}, B ={students failed physics}, C= {students failed both mathematics and physics}

Write your python code to compute

1) If the student failed physics, what is the probability that he failed mathematics? P(A|B)
2) If the student failed mathematics, what is the probability that he failed the physics? P(B|A)
3) What is the probability that he failed mathematics or physics? P(A∪ B)

5. Naïve Bayes Classification

Naïve Bayes models are a group of simple classification algorithms. They rely on Bayes' theorem, which is given by

$P(A|B) = \frac{P(A)P(B|A)}{P(B)}$

where $P(A|B)$ is called the posterior probability; $P(A)$ the prior; and $P(B|A)$ the likelihood. Suppose you are interested in finding the probability of a label (L) for a data given some observed features. Based on Bayes' theorem, we have

$$P(L|features) = \frac{P(features|L)P(L)}{P(features)}$$

If you are trying to decide between two labels (L_1 *and* L_2), then one way to make the decision is to compute the ratio of the posterior probabilities for each label, that is:

$$\frac{P(L_1|feature)}{P(L_2|feature)} = \frac{P(features|L_1)P(L_1)}{P(features|L_2)P(L_2)}$$

Naïve Bayes models may have different naïve assumptions about the data. In this section, you will focus on the Gaussian Naïve Bayes model.

First, import necessary modules and generate 100 data points with two classes as follows:

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
```

```
from sklearn.datasets import make_blobs
X, y = make_blobs(100, 2, centers=2, random_state=2, cluster_std=1.5)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='RdBu');
```

Note that we assume that the data has a Gaussian distribution with no covariance between two features. To find more details about the function **make_blobs()**, type **?make_blobs**.

Generate a model using the Naïve Bayes classifier in the following steps:

```
"""Import Gaussian Naive Bayes model"""
from sklearn.naive_bayes import GaussianNB

"""Create a Gaussian Naive Bayes classifier"""
model = GaussianNB()

"""Train the model using the training set"""
model.fit(X, y);
```

The model is fit by simply finding the mean and standard deviation of the points within each label, which are all you need to define a Gaussian distribution. With the distribution in place for each class, the likelihood $P(features|L)$ can computed for any data point. Thus, the posterior ratio can be computed, and which label is the most probable for a given point can be determined.

Now let's generate 1000 new data points and plot this new data (without the label information) together with the training dataset as follows:

```
rng = np.random.RandomState(0)
Xnew = [-6, -14] + [14, 18] * rng.rand(1000, 2)

plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='RdBu')
lim = plt.axis()
plt.scatter(Xnew[:, 0], Xnew[:, 1], s=20, cmap='RdBu', alpha=0.1)
plt.axis(lim);
```

Run your code to see what you have obtained.

To do the prediction, you need to type the following:

```
ynew = model.predict(Xnew)
```

To check the posterior probabilities of the first and second label, respectively, you may type

```
yprob = model.predict_proba(Xnew)
yprob[-8:].round(2)
```

Task7:

Plot the new data again with predicted label information. That is to use two different colours in terms of their label information to distinguish them.

Task8: Classifying the wine dataset using Naïve Bayes Classification

1) Load the data as follows:

```
from sklearn import datasets

wine = datasets.load_wine()
print("Feature Names:", wine.feature_names)
print("Labels:", wine.target_names)
```

2) Using proper Python command to check the size of the dataset, that is, how many data points are there? How many features are there?

3) Splitting the dataset into a training set (60%) and a test set (40%)
   a. Import train_test_split function as follows:

```
from sklearn.model_selection import train_test_split
```

4) Train a Gaussian Naïve Bayes model on the training dataset
5) Predict the label information using the trained model on the test dataset.
6) Report the accuracy rate on the test set using the following code:

```
#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics

# Model Accuracy
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

6. A bit more about Python programming – Object Oriented Programming

In this section, you will learn what object-oriented programming is and how you can write your own classes and create objects from them.

Let's have a look at the following code together:

```
class Student:
    """contents of the class"""
    def __init__(self, pModule, pName, pMark):
        self._module = pModule
        self.name = pName
        self.mark = pMark
        print('Creating Student Object')

    def __str__(self):
        return "Module = %s, Name = %s, Mark = %d" %(self.module, self.name, self.mark)

    def totalMark(self):
        prompt = '\nEnter the mark of the first coursework %s:' %(self.name)
        mark1 = input(prompt)
        prompt = '\nEnter the mark of the second coursework %s:' %(self.name)
        mark2 = input(prompt)
        self.mark = int(mark1)+int(mark2)
        return self.mark
```

A class is a template for grouping related data and methods together. For example, class Student can be used to store the name and registered module of the student.

In the example above, a class named Student is created by using the key word class.

Next, two special method called __init__ and __str__ for the class are defined. __init__ is known as the initializer of the class. It is always name init with two underscores in from and at the back. All special methods have two underscores in front and at the back of their names. __str__ is another special method that is commonly included to return a human readable string that represents the class.

In class Student, there are three variables, module, name and mark. These variables are known as instance variables, which are variables that are prefixed with a self keyword.

In addition, a method called **totalMark()** to calculate the total mark of the student is defined in the class. A method is very similar to a function, except for the parameter self. In fact, a method is almost identical to a function except that a method exists inside a class and most methods have self as a parameter.

As can be seen in the method of **totalMark()**, there is no the self keyword in front of some variables, such as, prompt, mark1 and mark2. This is because these variables are local variables and only exist within the method of **totalMark()**.

Task9: download the file called OOPI_Student.ipynb from Canvas.

1) Instantiate a Student object by something similar to the following:

```
student1 = Student('7COM1000', 'Jane', 0)
```

2) Access the variable name and module as follows:

```
student1.name
```

```
student1.module
```

3) You may change the code of module as follows:

```
student1.module = '7COM3000'
```

4) To use **totalMark()** method, you may type:

```
student1.totalMark()
```

5) Print the variable mark

References

1. Amit Saha: Doing math with Python: use programming to explore algebra, statistics, calculus and more!
2. Alberto Boschetti and Luca Massaron: Python data science essentials: become an efficient data science practitioner by thoroughly understanding the key concepts of Python. 2015 Packt Publishing
3. Jake VanderPlas: Python Data Science Handbook, 2016. O'Relly Media, Inc