Assignment II: Statistics and Plotting + Digital Image Processing

Introduction to Machine Learning Lab (190.013), SS2023

Björn Ellensohn[1]

[1]*m01435615, bjoern.ellensohn@stud.unileoben.ac.at, Montanuniversität Leoben, Austria*

April 27, 2023

This document guides through the process of solving Assignment 3.

1 Introduction

The Task of 3rd assignment was coming up with an abstract class called `ContinuousDistribution` that provides the outline for two subclasses `GaussDistribution` and `BetaDistribution`. Again, provided was a .csv file with a dataset. The assignment is divided into three main parts and the bonus part.

2 Part I - Abstract Class

2.1 Preparation

When searching online for information on Python's abstract classes, you may come across the module known as `ABC`. This module includes a specific decorator, `@abstractmethod`, that can be used to define abstract methods. Essentially, this allows you to create a class that serves as a template or blueprint for other classes, without actually implementing the methods. To create a functional child class, you must define and implement the necessary methods there.

2.2 Defining the Abstract Class

The class `ContinuousDistribution` is created that provides the basic outline for the main classes of this exercise. The following functions should be inlcuded:

- Data Import and Export using csv files.
- Computation of the mean based on the samples from the csv.
- Computation of the standard deviation based on the samples from the csv.
- Visualization of the distribution, the raw data or the generated samples.
- Generating/Drawing Samples from the distribution.

So in the end, the abstract class `ContinuousDistribution` should look like this:

```
import abc

class
↪  ContinuousDistribution(metaclass=abc.ABCMeta):
@abc.abstractmethod
def import_data(self, file_path):
    pass

@abc.abstractmethod
def export_data(self, data, file_path):
    pass

@abc.abstractmethod
def compute_mean(self, data):
    pass

@abc.abstractmethod
def compute_standard_deviation(self, data):
    pass

@abc.abstractmethod
```

```
def visualize(self, data=None):
    pass

@abc.abstractmethod
def generate_samples(self, n_samples):
    pass
```

3 Part II - Plot and Sample Gaussian Distributions

In the second part of this exercise, it is required to initiate a child class which is responsible for dealing with Gaussian distributions.

Explicitly, the following should be implemented:

- Implement the functions defined in “ContinousDistribution”.
- Implement a constructor that optionally takes the dimension of the multivariate distribution.
- Implement a visualization for Multivariate Gaussians up to 3 dimensions.
- Find the empirical parameters of the distribution that created the samples in the ‘MGD.csv’ file.
- Plot the samples of the ‘MGD.csv’ file and the sample from the learned distribution in two subfigures.

So the main goal of this class is to be able to compute the statistical parameters from a provided dataset and being able to sample a new dataset from this distribution. In the end, the datapoints should be plotted.

At first, let us declare how the Gaussian dsitribution is defined. For all normal distributions, 68.2% of the observations will appear within plus or minus one standard deviation of the mean; 95.4% of the observations will fall within +/- two standard deviations; and 99.7% within +/- three standard deviations. This fact is sometimes referred to as the "empirical rule," a heuristic that describes where most of the data in a normal distribution will appear. This means that data falling outside of three standard deviations ("3-sigma") would signify rare occurrences.

In mathematical terms this is explained in Equation 1.

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

To sample from a multivariate Gaussian distribution, I will use a module called `multivariate_normal` from the `scipy` package.

The constructor of `GaussianDistribution` then looks like this:

```
class GaussDistribution(ContinuousDistribution):
    def __init__(self, dim=1):
        self.dim = dim
        self.mean = np.zeros(dim)
        self.covariance = np.eye(dim)
        self.data = pd.DataFrame()
        self.samples = None
```

Figure 1 to Figure 3 are showing the end results. In the appendix you will find the full code.

Figure 1: *Plot of one dimensional Gaussian distribution.*

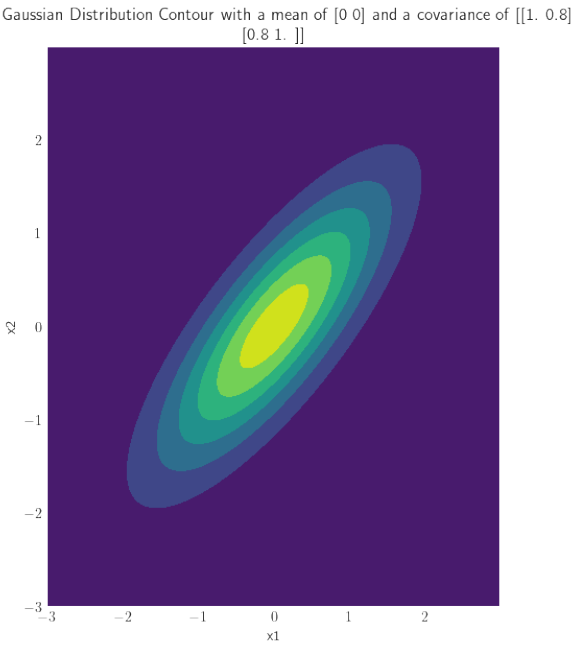

Figure 2: *Contour plot of two dimensional Gaussian distribution.*

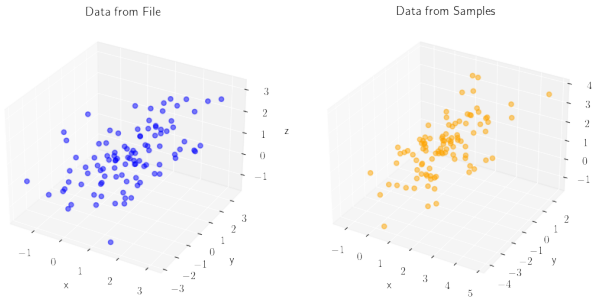

Figure 3: *Plots of three dimensional Gaussian distribution.*

4 Part III - Plot and Sample Beta Distributions

Next, a class `BetaDistribution` shouldbe inherited from the meta class. It should implement:

- Generate beta distributed samples and plot the distribution giving the parameters a and b.
- The constructor should take the parameters a and b as arguments.
- A visualization for Beta distributions, including the mean and the standard deviation lines.

Again, there is a module in the `scipy` package that does the hard work here `beta`. The constructor of `BetaDistribu` then looks like this:

```
class
↪  BetaDistribution(ContinuousDistribution):
def __init__(self, a, b):
    self.a = a
    self.b = b
    self.data = None
```

Provided is a plot showing the end results in Figure 4.

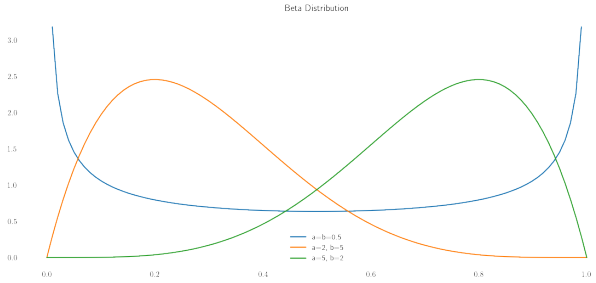

Figure 4: *Plot of three different Beta Distributions.*

APPENDIX

Ladies and Gentlemen, here I introduce the code. And figures.
Questions to: bjoern.ellensohn@stud.unileoben.ac.at

Figures