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 - `ContinuousDistribution`

2.1 Preparation

Provided .csv file needs to be imported into Python somehow. There is a library called csv dealing with this job. Sadly, the data object created by the `csv` library does not properly recognize the formatting of the .csv file. Hence, there is some reformatting applied afterwards. The data is saved into a data list.

```
with open('data.csv', newline='') as f:
reader = csv.reader(f)
data = []
for row in reader:
  data.append(', '.join(row))
```

2.2 Defining Statistics class

The class `BasicStastistics` is created that provides some functions to analyze the dataset. Not all of it is shown here.

```
import math

class BasicStatistics:
    def __init__(self, data):
        self.data = data

    def mean(self):
        n = len(self.data)
        total = sum(float(x) for x in self.data)
        return total/n
```

To apply the statistical methods, the BasicStatistics class must be initialized with the dataset. Then the functions can be called as usual.

```
bs = BasicStatistics(data)
print("Mean:", bs.mean())
print("Median:", bs.median())
print("Variance:", bs.variance())
print("Standard deviation:",
↪  bs.standard_deviation())
print("Normalized data:", bs.normalize())
```

2.3 Plotting

The most difficult passage of the first part was plotting the data. Using pyplot, you can use subplots to include several plots into a single figure and align them to your liking. But especially the alignment was kind of frustrating. At last, I ended up with a quite nice result which can be rather

informative. However, before plotting, the dataset had to be normalized and included into a new BasicStastistics-object.

```
normalized_data = bs.normalize()
norm = BasicStatistics(normalized_data)
```

Subplot 1 should the original dataset displayed as a histogram, overlaid with the mean, median and standard deviation. For subplot 2 and 3, the comparison of the data points between the original dataset and the normalized dataset should be emphasized. That said, arranging the plot took place. Here, 211, 223, and 224 indicate the position of the subplots in the 2 by 2 grid.

```
fig = plt.subplots(nrows=2, ncols=2,
↪  figsize=(20, 15))
ax1 = plt.subplot(211)
ax2 = plt.subplot(223)
ax3 = plt.subplot(224)
```

The end result can be seen Figure **??**.

3 Part II - Fun with Matrices

The second part allowed using Numpy as a little helper for doing matrix calculus.

Numpy is a powerful tool for playing with matrices, already providing many operations in the library which comes in handy. For example, creating a matrix goes as follows:

```
matrix = np.random.rand(10, 50)
```

This returns a matrix consisting of random numbers with the dimensions 10 x 50. Then the matrix object already provides several matrix manipulations as part of their attributes. For instance, calling the shape of the matrix looks like this:

```
matrix.shape
```

The rest of the assignment was solved using the already implemented functions. Here I provide a list of them:

- Shape of a matrix - `matrix.shape`
- Dot product of matrices a and b - `np.dot(a, b)`
- Power p of matrix D - `np.linalg.matrix_power(D, p)`
- Determinant of matrix E - `np.linalg.det(E)`
- Inverse of matrix F - `np.linalg.inv(F)`
- Pseudoinverse of matrix G - `np.linalg.pinv(G)`
- Eigenvalues and Eigenvectors of matrix H - `np.linalg.eig(H)`

In the appendix you will find the full code.

4 Part III - Bonus Part

In the last section I will describe the bonus point assignment. I chose the image processing one. To start we got an unsolved Jupyter notebook which guides through the task. The aim was to apply some image processing algorithms to an image of a robot hand.

At first, I replaced the image uploading part by including an import function for calling the image by its URL:

```
# Get Image:
from urllib.request import urlopen

# URL of the image to download
url =
↪  "https://cps.unileoben.ac.at/wp/DALL%C2%B7E-2023-02-0
# Download the image and open it with Pillow
with urlopen(url) as response:
    image = Image.open(response)

# Save image as file
image.save("hand.png")
```

Although the notebook included instructions for every part, all the functions in the assignment where just empty, so they had to be filled with the appropriate code. The first function should return the filtered image, taking the kernel and the raw image as arguments. The steps this function has to caluclate are as follows:

- Calculate offset for kernel
- Create zero-filled array for output
- Iterate over all pixel coordinates to extract kernel-sized sub-image from input image and perform element-wise multiplication between sub-image and kernel, and sum up the results
- Also clipping is needed to stay inside the maximum values of 255
- Finally storing those values in an array

The second function implemented a way of calling function 1, providing the parameters and returning the converted output from function 1.

The main function initializes the Laplacian and the Gaussian kernel as transformation matrices.

In the end, the Gaussian and the Laplacian processing where demonstrated. The Gaussian can be seen in Figure **??** and the Laplacian in Figure **??**.

APPENDIX

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

Figures