Term: L3-AII & L3-ElnI

Semester: 5

AY: 2023-2024

Abdelbacet Mhamdi

Dr.-Ing. in Electrical Engineering

Senior Lecturer at ISET Bizerte

abdelbacet.mhamdi@bizerte.r-iset.tn

Machine Learning

LAB MANUAL

Institut Supérieur des Études Technologiques de Bizerte

Available @ https://github.com/a-mhamdi/mlpy/

Honor code

THE UNIVERSITY OF NORTH CAROLINA AT CHAPEL HILL
Department of Physics and Astronomy
http://physics.unc.edu/undergraduate-program/labs/general-info/

"During this course, you will be working with one or more partners with whom you may discuss any points concerning laboratory work. However, you must write your lab report, in your own words.

Lab reports that contain identical language are not acceptable, so do not copy your lab partner's writing.

If there is a problem with your data, include an explanation in your report. Recognition of a mistake and a well-reasoned explanation is more important than having high-quality data, and will be rewarded accordingly by your instructor. A lab report containing data that is inconsistent with the original data sheet will be considered a violation of the Honor Code.

Falsification of data or plagiarism of a report will result in prosecution of the offender(s) under the University Honor Code.

On your first lab report you must write out the entire honor pledge:

The work presented in this report is my own, and the data was obtained by my lab partner and me during the lab period.

On future reports, you may simply write "*Laboratory Honor Pledge*" and sign your name."

Contents

In order to activate the virtual environment and launch **Jupyter Notebook**, we recommend you to proceed as follow

① Press simultaneously the keys & R on the keyboard. This will open the dialog box `Run`;

② Then enter `cmd` in the command line and confirm with ↵ key on the keyboard;

③ Type the instruction `mlpy.bat` in the console prompt line;

Command Prompt

```
C:\Users\admin> mlpy.bat
```

④ Finally press the ↵ key.

Leave the system console active.

1 | *Python* Onramp

Student's name	. . . .	. . . .	. . . .
Score ***/20***	. . . .	. . . .	. . . .
	Detailed Credits		
Anticipation ***(4 points)***	. . . .	. . . .	. . . .
Management ***(2 points)***	. . . .	. . . .	. . . .
Testing ***(7 points)***	. . . .	. . . .	. . . .
Data Logging ***(3 points)***	. . . .	. . . .	. . . .
Interpretation ***(4 points)***	. . . .	. . . .	. . . .

Motivations

- ⋆ *Python* is a popular programming language in the field of machine learning because it is relatively easy to learn and has a wide range of libraries and frameworks that support machine learning tasks.
- ⋆ *Python* has a large and active community of developers, which means that there are many resources available online, such as tutorials, documentation, and online forums, to help us learn and troubleshoot our code.
- ⋆ Many machine learning tools and frameworks, such as *TensorFlow* and *scikit-learn*, are written in *Python*, which makes it easy to integrate these tools into *Python* programs.
- ⋆ *Python* is a versatile language that can be used for a wide range of applications beyond machine learning, including web development, data analysis, and scientific computing. Learning *Python* can therefore open up many career opportunities for us.

The notebook is available at `https://github.com/a-mhamdi/mlpy/` ⟶ *Codes* ⟶ *Python* ⟶ *py-onramp.ipynb*

Goals

1. Learn the basics of programming in *Python*;
2. Get familiar with *Jupyter Notebook*;
3. Use the modules of scientific computing.

1.1 Numerical variables & types

[1]:
```
a = 1 # An integer
print('The variable a = {} is of type {}'.format(a, type(a)))
```

```
The variable a = 1 is of type <class 'int'>
```

[2]:
```
b = -1.25 # A floating number
print('The variable b = {} is of type {}'.format(b, type(b)))
```

```
The variable b = -1.25 is of type <class 'float'>
```

[3]:
```
c = 1+0.5j # A complex number
print('The variable c = {} is of type {}'.format(c, type(c)))
```

```
The variable c = (1+0.5j) is of type <class 'complex'>
```

1.2 Strings

[4]:
```
msg = "My 1st lab!"
print(msg, type(msg), sep = '\n***\n') # \n: Carriage Return & Line Feed
print(msg + 3* '\nPython is awesome')
```

```
My 1st lab!
***
<class 'str'>
My 1st lab!
Python is awesome
Python is awesome
Python is awesome
```

[5]:
```
longMsg = """This is a long message,
spanned over multiple lines"""
print(longMsg)
```

```
This is a long message,
spanned over multiple lines
```

Indexing and slicing

[6]:
```
# Positive indexing
print(msg, msg[1:5], sep = ' -----> ')
# Negative indexing
print(msg, msg[-5:-1], sep = ' -----> ')
```

```
My 1st lab! -----> y 1s
My 1st lab! ----->  lab
```

String transformations

[7]:
```
msg = 'A message'
print(len(msg))
```

```
print(msg.lower())
print(msg.upper())
print(msg.split(' '))
print(msg.replace('mes', 'MES'))
print('a' in msg) #  Check if the variable `msg` contains the letter 'a'
```

```
9
a message
A MESSAGE
['A', 'message']
A MESSage
True
```

[8]:
```
price, number, perso = 300, 7, 'A customer'
print('{} asks for {} pieces. They cost {} TND!'.format(perso, number,␣
 ↪price))
print('{1} demande {2} pièces. They cost {0} TND!'.format(price, perso,␣
 ↪number))
```

```
A customer asks for 7 pieces. They cost 300 TND!
A customer demande 7 pièces. They cost 300 TND!
```

1.3 Binary, octal & hexadecimal

[9]:
```
x = 0b0101 # 0b : binary
print(x, type(x), sep = '\t----\t') # \t : tabular
y = 0xAF # 0x : hexadecimal
print(y, type(y), sep = '\t' + '---'*5 + '\t')
z = 0o010 # 0o : octal
print(z, type(z), sep = ', ')
```

```
5       ----    <class 'int'>
175     --------------- <class 'int'>
8, <class 'int'>
```

Boolean

[10]:
```
a = True
b = False
print(a)
print(b)
```

```
True
False
```

[11]:
```
print("50 > 20 ? : {} \n50 < 20 ? : {} \n50 = 20 ? : {}\n50 /= 20 ? : {}"
      .format(50 > 20, 50 < 20, 50 == 20, 50 != 20)
     )
```

```
50 > 20 ? : True
50 < 20 ? : False
```

```
50 = 20 ? : False
50 /= 20 ? : True
```

[12]:
```python
print(bool(123), bool(0), bool('Lab'), bool())
```

```
True False True False
```

[13]:
```python
var1 = 100
print(isinstance(var1, int))
var2 = -100.35
print(isinstance(var2, int))
print(isinstance(var2, float))
```

```
True
False
True
```

1.4 Lists, tuples & dictionaries

In Python, a list is an ordered collection of items that can be of any data type (including other lists). Lists are defined using square brackets, with items separated by commas. For example:

[14]:
```python
shopping_list = ['milk', 'eggs', 'bread', 'apples']
```

A tuple is also an ordered collection of items, but it is immutable, meaning that the items it contains cannot be modified once the tuple is created. Tuples are defined using parentheses, with items separated by commas. For example:

[15]:
```python
point = (3, 5)
```

A dictionary is a collection of key-value pairs, where the keys are unique and used to look up the corresponding values. Dictionaries are defined using curly braces, with the key-value pairs separated by commas. The keys and values are separated by a colon. For example:

[16]:
```python
phonebook = {'Alice': '555-1234', 'Bob': '555-5678', 'Eve': '555-9101'}
```

You can access the items in a list or tuple using an index, and you can access the values in a dictionary using the corresponding keys. For example:

[17]:
```python
# Accessing the second item in a list
print(shopping_list[1])  # prints 'eggs'

# Accessing the first item in a tuple
print(point[0])  # prints 3

# Accessing the phone number for 'Bob' in the phonebook dictionary
print(phonebook['Bob'])  # prints '555-5678'
```

```
eggs
3
555-5678
```

1.4.1 List

[18]:
```
lst = ['a', 'b', 'c', 1, True] # An aggregate of various types
print(lst)
```

```
['a', 'b', 'c', 1, True]
```

[19]:
```
print(len(lst)) # Length of `lst` variable
print(lst[1:3]) # Accessing elements of `lst`
lst[0] = ['1', 0] # Combined list
print(lst)
print(lst[3:])
print(lst[:3])
```

```
5
['b', 'c']
[['1', 0], 'b', 'c', 1, True]
[1, True]
[['1', 0], 'b', 'c']
```

[20]:
```
lst.append('etc') # Insert 'etc' at the end
print(lst)
```

```
[['1', 0], 'b', 'c', 1, True, 'etc']
```

[21]:
```
lst.insert(1, 'xyz') # Inserting 'xyz'
print(lst)
```

```
[['1', 0], 'xyz', 'b', 'c', 1, True, 'etc']
```

[22]:
```
lst.pop(1)
print(lst)
```

```
[['1', 0], 'b', 'c', 1, True, 'etc']
```

[23]:
```
lst.pop()
print(lst)
```

```
[['1', 0], 'b', 'c', 1, True]
```

[24]:
```
del lst[0]
print(lst)
```

```
['b', 'c', 1, True]
```

[25]:
```
lst.append('b')
print(lst)
lst.remove('b')
print(lst)
```

```
['b', 'c', 1, True, 'b']
['c', 1, True, 'b']
```

[26]:
```
# Loop
for k in lst:
    print(k)
```

```
c
1
True
b
```

[27]:
```
lst.clear()
print(lst)
```

```
[]
```

Method	*Description*
copy()	Returns a copy of the list
list()	Transforms into a list
extend ()	Extends a list by adding elements at its end
count()	Returns the occurrences of the specified value
index()	Returns the index of the first occurrence of a specified value
reverse()	Reverse a list
sort()	Sort a list

1.4.2 Tuples

[28]:
```
tpl = (1, 2, 3)
print(tpl)
```

```
(1, 2, 3)
```

[29]:
```
tpl = (1, '1', 2, 'text')
print(tpl)
```

```
(1, '1', 2, 'text')
```

[30]:
```
print(len(tpl))
```

```
4
```

[31]:
```
print(tpl[1:])
```

```
('1', 2, 'text')
```

[32]:
```
try:
    tpl.append('xyz') # Throws an error
except Exception as err:
    print(err)
```

```
'tuple' object has no attribute 'append'
```

[33]:
```
try:
    tpl.insert(1, 'xyz') # Throws an error
except Exception as err:
    print(err)
```

```
'tuple' object has no attribute 'insert'
```

[34]:
```
my_lst = list(tpl)
my_lst.append('xyz')
print(my_lst, type(my_lst), sep = ', ')
```

```
[1, '1', 2, 'text', 'xyz'], <class 'list'>
```

[35]:
```
nv_tpl = tuple(my_lst) # Convert 'my_lst' into a tuple 'nv_tpl'
print(nv_tpl, type(nv_tpl), sep = ', ')
```

```
(1, '1', 2, 'text', 'xyz'), <class 'tuple'>
```

[36]:
```
# Loop
for k in nv_tpl:
    print(k)
```

```
1
1
2
text
xyz
```

[37]:
```
rs_tpl = tpl + nv_tpl
print(rs_tpl)
```

```
(1, '1', 2, 'text', 1, '1', 2, 'text', 'xyz')
```

1.4.3 Dictionaries

[38]:
```
# dct = {"key": "value"}
dct = {
    "Term" : "GM",
    "Speciality" : "ElnI",
    "Sem" : "4"
}
print(dct, type(dct), sep = ', ')
```

```
{'Term': 'GM', 'Speciality': 'ElnI', 'Sem': '4'}, <class 'dict'>
```

[39]:
```
print(dct["Sem"])
sem = dct.get("Sem")
print(sem)
```

```
4
4
```

[40]:
```
dct["Term"] = "GE"
print(dct)
```

```
{'Term': 'GE', 'Speciality': 'ElnI', 'Sem': '4'}
```

[41]:
```
# Loop
for d in dct:
    print(d, dct[d], sep = '\t|\t')
```

```
Term    |       GE
Speciality      |       ElnI
Sem     |       4
```

[42]:
```
for k in dct.keys():
    print(k)
```

```
Term
Speciality
Sem
```

[43]:
```
for v in dct.values():
    print(v)
```

```
GE
ElnI
4
```

1.5 NumPy

NumPy is a *Python* library that is used for scientific computing and data analysis. It provides support for large, multi-dimensional arrays and matrices of numerical data, and a large library of mathematical functions to operate on these arrays.

One of the main features of *NumPy* is its *N*-dimensional array object, which is used to store and manipulate large arrays of homogeneous data (*i.e.*, data of the same type, such as integers or floating point values). The array object provides efficient operations for performing element-wise calculations, indexing, slicing, and reshaping.

NumPy also includes a number of functions for performing statistical and mathematical operations on arrays, such as mean, standard deviation, and dot product. It also includes functions for linear algebra, random number generation, and Fourier transforms.

Official documentation can be found at https://numpy.org/

[44]:
```
import numpy as np
```

NumPy vs List

[45]:
```
a_np = np.arange(6) # NumPy
print("a_np = ", a_np)
```

```
print(type(a_np))
a_lst = list(range(0,6)) # List
print("a_lst = ", a_lst)
print(type(a_lst))
# Comparison
print("2 * a_np = ", a_np * 2)
print("2 * a_lst = ", a_lst * 2)
```

```
a_np =  [0 1 2 3 4 5]
<class 'numpy.ndarray'>
a_lst =  [0, 1, 2, 3, 4, 5]
<class 'list'>
2 * a_np =  [ 0  2  4  6  8 10]
2 * a_lst =  [0, 1, 2, 3, 4, 5, 0, 1, 2, 3, 4, 5]
```

[46]:
```
v_np = np.array([1, 2, 3, 4, 5, 6]) # NB : parentheses then brackets, i.e,␣
↪([])
print(v_np)
```

```
[1 2 3 4 5 6]
```

[47]:
```
v_np = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
print(v_np)
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
```

[48]:
```
print(type(v_np))
```

```
<class 'numpy.ndarray'>
```

[49]:
```
print(v_np[0])
```

```
[1 2 3 4]
```

[50]:
```
v_np.ndim # Dimensions of v_np
```

[50]: 2

[51]:
```
v_np.shape # Number of lignes and columns, may be more
```

[51]: (3, 4)

[52]:
```
v_np.size # How many elements are in `v_np`
```

[52]: 12

If we need to create a matrix $(3,3)$, we can do as follows:

[53]:
```
u = np.arange(9).reshape(3,3)
print(u)
```

```
[[0 1 2]
 [3 4 5]
 [6 7 8]]
```

Let us see some known operations to do on matrices

[54]:
```
M = np.array([[1, 2], [1, 2]])
print(M)
```

```
[[1 2]
 [1 2]]
```

[55]:
```
N = np.array([[0, 3], [4, 5]])
print(N)
```

```
[[0 3]
 [4 5]]
```

Addition

[56]:
```
print(M + N)
print(np.add(M, N))
```

```
[[1 5]
 [5 7]]
[[1 5]
 [5 7]]
```

Subtraction

[57]:
```
print(M-N)
print(np.subtract(M, N))
```

```
[[ 1 -1]
 [-3 -3]]
[[ 1 -1]
 [-3 -3]]
```

Element-wise Product

Element-wise multiplication, also known as **Hadamard product**, is an operation that multiplies each element of one matrix with the corresponding element of another matrix. It is denoted by the symbol $\odot$ or `.*` in some programming languages.

For example, consider the following matrices:

$$A = \begin{bmatrix} a_1, & a_2, & a_3 \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} b_1, & b_2, & b_3 \end{bmatrix}$$

The element-wise product of these matrices is:

$$A \odot B = \begin{bmatrix} a_1 b_1, & a_2 b_2, & a_3 b_3 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 \\ 1 & 2 \end{bmatrix} . \times \begin{bmatrix} 0 & 3 \\ 4 & 5 \end{bmatrix} = \begin{bmatrix} 0 & 6 \\ 4 & 10 \end{bmatrix}$$

We need element-wise multiplication in many applications. For example, in image processing, element-wise multiplication is used to modify the intensity values of an image by multiplying each pixel value with a scalar value. In machine learning, element-wise multiplication is used in the implementation of various neural network layers, such as convolutional layers and fully connected layers. Element-wise multiplication is also used in many other mathematical and scientific applications.

[58]:
```
print(M * N)
print(np.multiply(M, N))
```

```
[[ 0  6]
 [ 4 10]]
[[ 0  6]
 [ 4 10]]
```

Dot Product

$$\begin{bmatrix} 1 & 2 \\ 1 & 2 \end{bmatrix} \times \begin{bmatrix} 0 & 3 \\ 4 & 5 \end{bmatrix} = \begin{bmatrix} 8 & 13 \\ 8 & 13 \end{bmatrix}$$

[59]:
```
print(M.dot(N))
print(np.dot(M, N))
```

```
[[ 8 13]
 [ 8 13]]
[[ 8 13]
 [ 8 13]]
```

Element-wise Division

$$\begin{bmatrix} 0 & 3 \\ 4 & 5 \end{bmatrix} ./ \begin{bmatrix} 1 & 2 \\ 1 & 2 \end{bmatrix} = \begin{bmatrix} 0:1 & 3:2 \\ 4:1 & 5:2 \end{bmatrix}$$

[60]:
```
print(N / M)
print(np.divide(N, M))
```

```
[[0.  1.5]
 [4.  2.5]]
[[0.  1.5]
 [4.  2.5]]
```

Determinant of a matrix

[61]:
```
print("Determinant of M:")
print(np.linalg.det(M))
print("Determinant of N:")
print(np.linalg.det(N))
```

```
Determinant of M:
0.0
Determinant of N:
-12.0
```

1.6 Matplotlib

Matplotlib is a 2D data visualization library in *Python* that allows users to create a wide range of static, animated, and interactive visualizations in *Python*. It is one of the most widely used data visualization libraries in the *Python* data science ecosystem and is particularly useful for creating line plots, scatter plots, bar plots, error bars, histograms, bar charts, pie charts, box plots, and many other types of visualizations.

Matplotlib is built on top of *NumPy* and is often used in conjunction with other libraries in the PyData ecosystem, such as *Pandas* and *Seaborn*, to create complex visualizations of data. It is also compatible with a number of different backends, such as the *Jupyter notebook*, *Qt*, and *Tkinter*, which allows it to be used in a wide range of environments and contexts.

The full documentation and an exhaustive list of samples can be found at https://matplotlib.org/

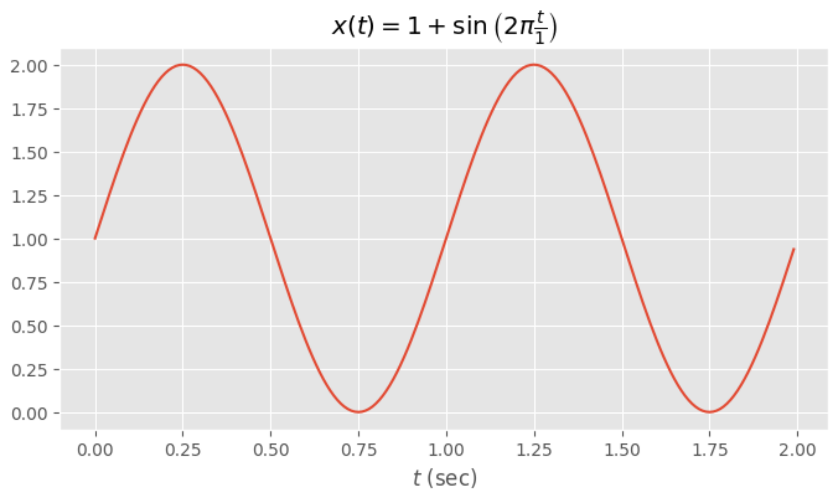
[62]:
```
import numpy as np
from matplotlib import pyplot as plt

plt.style.use("ggplot")
plt.rcParams['figure.figsize'] = [8, 4]
```

We begin by creating a sinusoidal waveform denoted by x, period is 1 sec. The offset is 1.

[63]:
```
# Continuous function
t = np.arange(0.0, 2.0, 0.01)
x = 1 + np.sin(2 * np.pi * t) # Frequency = 1Hz
```

The set of instructions that allow to plot (x) are:

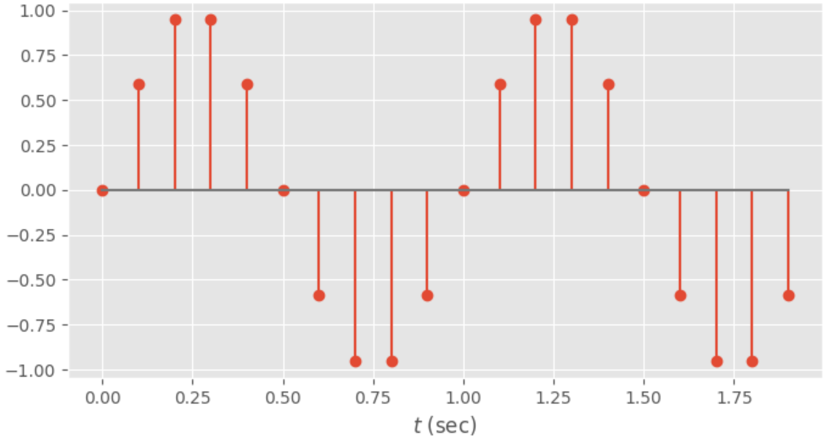
[64]:
```
plt.plot(t, x)

# Give the graph a title
plt.title(r"$x(t) = 1+\sin\left(2\pi\frac{t}{1}\right)$")
plt.xlabel("$t$ (sec)"); # Label the axis
```



[65]:
```
# Discret Function
t = np.arange(0.0, 2.0, 0.1)
y = np.sin(2*np.pi*t) # Same thing! Sinusoidal signal
```

[66]:
```
plt.stem(t, y)
plt.xlabel("$t$ (sec)")
```

[66]:
```
Text(0.5, 0, '$t$ (sec)')
```

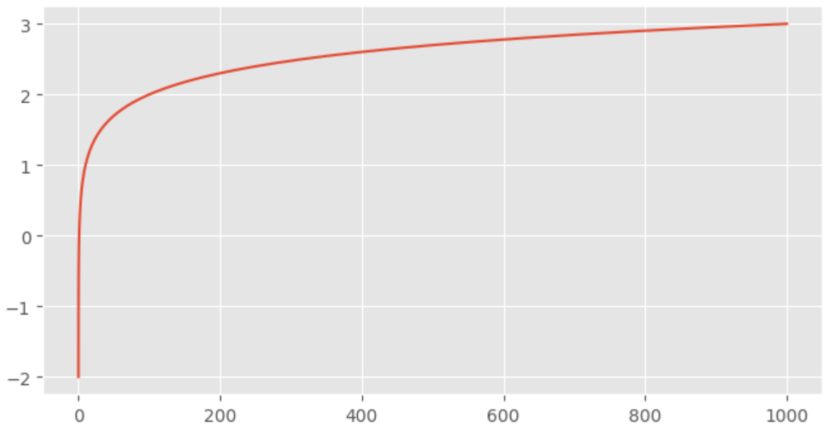


[67]:
```
x = np.logspace(-2, 3, 100)
y = np.log10(x)
```

[68]:
```
np.log10?
```

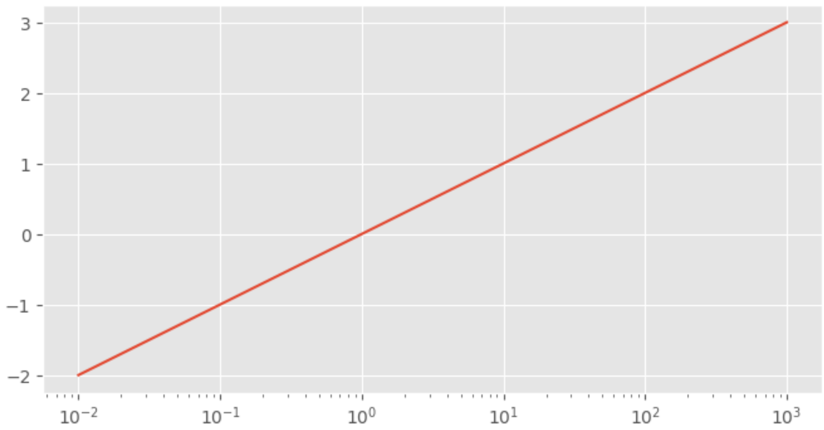
[69]:
```
plt.plot(x, y)
```

[69]:
```
[<matplotlib.lines.Line2D at 0x7f2199ca91e0>]
```



[70]:
```
plt.semilogx(x, y)
```

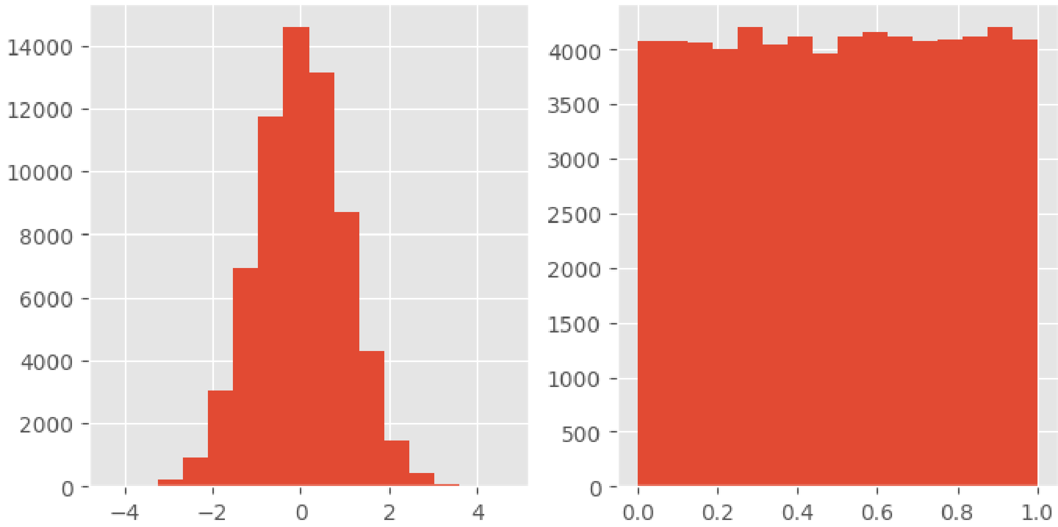
[70]:
```
[<matplotlib.lines.Line2D at 0x7f2157bb5a50>]
```



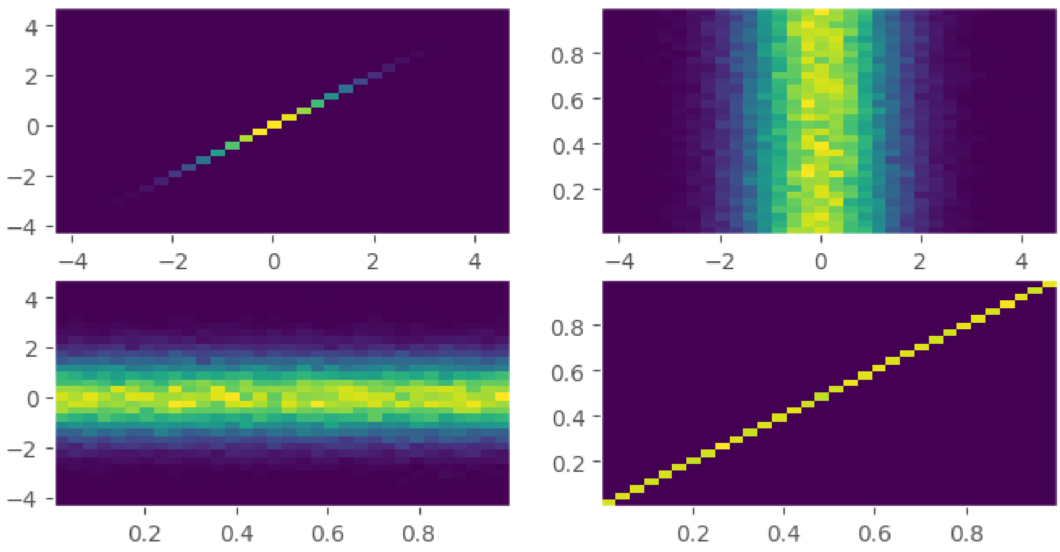
About distributions

[71]:
```
a = np.random.randn(2**16) # Normal Distribution
b = np.random.rand(2**16) # Uniform Distribution
```

[72]:
```
_, ax = plt.subplots(1, 2)
ax[0].hist(a, bins=16)
ax[1].hist(b, bins=16);
```



[73]:
```
_, ax = plt.subplots(2, 2)
ax[0][0].hist2d(a, a, bins=32)
ax[0][1].hist2d(a, b, bins=32)
ax[1][0].hist2d(b, a, bins=32)
ax[1][1].hist2d(b, b, bins=32);
```


4
2
0
−2
−4
−4 −2 0 2 4
0.8
0.6
0.4
0.2
−4 −2 0 2 4
4
2
0
−2
−4
0.2 0.4 0.6 0.8
0.8
0.6
0.4
0.2
0.2 0.4 0.6 0.8

2 Linear Regression

Student's name	. . . .	. . . .	. . . .
Score ***/20***	. . . .	. . . .	. . . .
		Detailed Credits	
Anticipation ***(4 points)***	. . . .	. . . .	. . . .
Management ***(2 points)***	. . . .	. . . .	. . . .
Testing ***(7 points)***	. . . .	. . . .	. . . .
Data Logging ***(3 points)***	. . . .	. . . .	. . . .
Interpretation ***(4 points)***	. . . .	. . . .	. . . .

Motivations

- ★ Linear regression is a fundamental statistical technique that is widely used in many fields, including economics, finance, biology, and computer science. It is a simple and effective way to model the relationship between a dependent variable and one or more independent variables.
- ★ Linear regression is relatively easy to understand and implement, making it a good starting point for us who are new to statistical modeling. It is also a good foundation for learning more advanced statistical techniques, such as multivariate or logistic regression.
- ★ Linear regression can be an useful tool for making predictions and understanding the underlying trends in data. It can help us to better understand and analyze data, and to make informed decisions based on our findings.

The notebook is available at https://github.com/a-mhamdi/mlpy/ → *Codes* → *Python* → *multiple-linear-regression.ipynb*

Multiple linear regression is a type of regression analysis in which there are multiple independent variables that have an effect on the dependent variable. In multiple linear regression, the goal is to find the linear equation that best explains the relationship between the outcome and the features in X.

The equation takes the form:

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_{m-1} x_{m-1}$$

where y is the dependent variable, $x_1, x_2, \ldots, x_{m-1}$ are the independent variables, and $\theta_0, \theta_1, \theta_2, \ldots, \theta_{m-1}$ are the coefficients that represent the influence of each variable on the output y. The coefficients are estimated using the data, and the resulting equation can be used later to make predictions on new data.

Importing the libraries

[1]:
```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
```

[2]:
```
np.set_printoptions(precision=3)
```

[3]:
```
# Show plots in an interactive format, e.g., zooming, saving, etc
%matplotlib inline
```

[4]:
```
plt.style.use('ggplot')
```

Importing the dataset

[5]:
```
df = pd.read_csv('./datasets/50_Startups.csv')
```

[6]:
```
df.head()
```

[6]:

	R&D Spend	Administration	Marketing Spend	State	Profit
0	165349.20	136897.80	471784.10	New York	192261.83
1	162597.70	151377.59	443898.53	California	191792.06
2	153441.51	101145.55	407934.54	Florida	191050.39
3	144372.41	118671.85	383199.62	New York	182901.99
4	142107.34	91391.77	366168.42	Florida	166187.94

[7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   R&D Spend        50 non-null     float64
 1   Administration   50 non-null     float64
 2   Marketing Spend  50 non-null     float64
 3   State            50 non-null     object
 4   Profit           50 non-null     float64
dtypes: float64(4), object(1)
memory usage: 2.1+ KB
```

[8]:
```
df.describe()
```

[8]:

	R&D Spend	Administration	Marketing Spend	Profit
count	50.000000	50.000000	50.000000	50.000000
mean	73721.615600	121344.639600	211025.097800	112012.639200
std	45902.256482	28017.802755	122290.310726	40306.180338
min	0.000000	51283.140000	0.000000	14681.400000

```
25%     39936.370000   103730.875000    129300.132500    90138.902500
50%     73051.080000   122699.795000    212716.240000   107978.190000
75%    101602.800000   144842.180000    299469.085000   139765.977500
max    165349.200000   182645.560000    471784.100000   192261.830000
```

Extract features *X* and target *y* from the dataset. **Profit** is the dependant variable.

[9]:
```
X = df.iloc[:, :-1]
y = df.iloc[:, -1]
```

Check the first five observations within *X*

[10]:
```
X.head()
```

[10]:
```
   R&D Spend  Administration  Marketing Spend       State
0  165349.20       136897.80        471784.10    New York
1  162597.70       151377.59        443898.53  California
2  153441.51       101145.55        407934.54     Florida
3  144372.41       118671.85        383199.62    New York
4  142107.34        91391.77        366168.42     Florida
```

[11]:
```
X = X.values
type(X)
```

[11]:
```
numpy.ndarray
```

Check the corresponding first five values from **Profit** column.

[12]:
```
y.head()
```

[12]:
```
0    192261.83
1    191792.06
2    191050.39
3    182901.99
4    166187.94
Name: Profit, dtype: float64
```

[13]:
```
y = y.values
type(y)
```

[13]:
```
numpy.ndarray
```

Encoding categorical data

[14]:
```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
```

[15]:
```
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [3])],␣
 ↪remainder='passthrough')
X = np.array(ct.fit_transform(X))
```

[16]:
```
print(X[:5])
```

```
[[0.0 0.0 1.0 165349.2 136897.8 471784.1]
 [1.0 0.0 0.0 162597.7 151377.59 443898.53]
 [0.0 1.0 0.0 153441.51 101145.55 407934.54]
 [0.0 0.0 1.0 144372.41 118671.85 383199.62]
 [0.0 1.0 0.0 142107.34 91391.77 366168.42]]
```

Splitting the dataset into training set and test set

[17]:
```
from sklearn.model_selection import train_test_split
```

[18]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=123)
```

Training the multiple linear regression model on the training set

[19]:
```
from sklearn.linear_model import LinearRegression
```

This code will create a linear regression model that fits a line to the training data, in order to make future predictions on the test data.

[20]:
```
lr = LinearRegression()
lr.fit(X_train, y_train)
```

[20]:
```
LinearRegression()
```

[21]:
```
theta = lr.coef_
theta
```

[21]:
```
array([-1.455e+02, -4.153e+02,  5.607e+02,  7.753e-01, -1.645e-02,
        3.627e-02])
```

[22]:
```
b = lr.intercept_
b
```

[22]:
```
48661.699896543345
```

Consider the sample `tst` as follows:

[23]:
```
tst = np.array([1, 0, 0, 15e+3, 10e+2, 5e+6])
```

Predict the outcome if `tst` is the input.

[24]:
```
tst = np.array([1, 0, 0, 15e+3, 10e+2, 5e+6])
pred = theta @ tst + b
print('%.3f' % pred)
```

```
241495.528
```

By calling our `lr`, we get the same result:

[25]:
```
lr.predict(tst.reshape(1, -1))
```

[25]:
```
array([241495.528])
```

If we don't want to do the encoding of state feature by ourselves, we can invoke the previous `ct` object.

[26]:
```
tst_new = [[15e+3, 10e+2, 5e+6, 'California']]
arr = np.array(ct.transform(tst_new))
arr
```

[26]:
```
array([[1.0, 0.0, 0.0, 15000.0, 1000.0, 5000000.0]], dtype=object)
```

[27]:
```
lr.predict(arr)
```

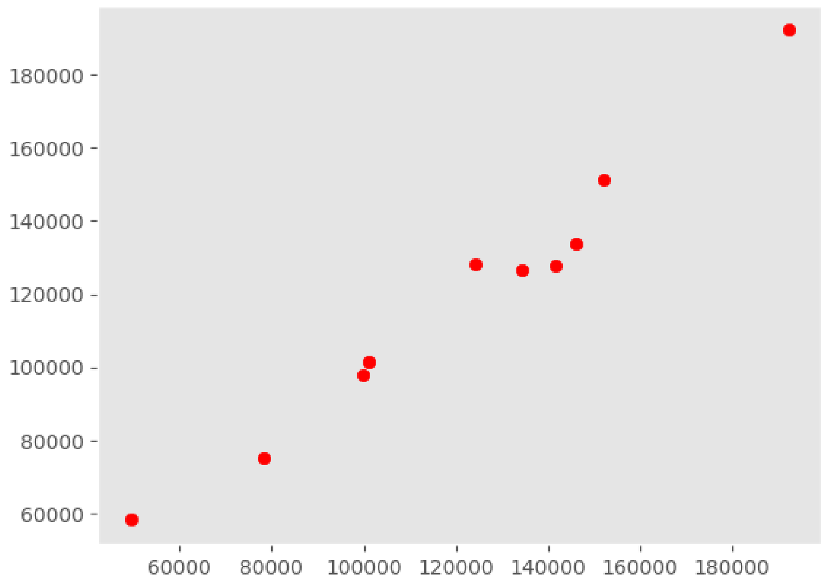
[27]:
```
array([241495.528])
```

Evaluation and Visualization

Make predictions using the X test set and visualize the results

[28]:
```
y_pred = lr.predict(X_test)
```

[29]:
```
plt.scatter(y_test, y_pred, c='red') # INSIGHT ON THE CORRELATION BETWEEN␣
↪Y_TEST AND Y_PRED
plt.grid()
```



Multiple linear regression can be used to understand the relationship between multiple independent variables and a single dependent variable, and can be used to make predictions about the dependent variable given new data. However, it's important to note that the independent variables must be linearly related to the dependent variable in order for multiple linear regression to behave appropriately. If the relationship is non-linear, we need to use a different type of regression analysis such as polynomial regression.

3 k-NN for Classification

Student's name	....	....	....
Score **/20**	....	....	....

Detailed Credits

Anticipation *(4 points)*	....	....	....
Management *(2 points)*	....	....	....
Testing *(7 points)*	....	....	....
Data Logging *(3 points)*	....	....	....
Interpretation *(4 points)*	....	....	....

Motivations

- ⋆ k-nearest neighbors (k-NN) is a simple and effective classification algorithm that is easy to understand and implement. It is based on the idea of using the class labels of the "nearest neighbors" to predict the class label of a new data point.
- ⋆ k-NN is a "lazy learner" that does not make any assumptions about the underlying data distribution, which makes it a good choice for working with complex or non-linear data. It is also robust to noise and can handle missing data. As a result, k-NN is often used as a baseline method for comparison with more advanced classification algorithms.

The notebook is available at `https://github.com/a-mhamdi/mlpy/` → *Codes* → *Python* → *k-nearest-neighbors.ipynb*

k-nearest neighbors (k-NN) is a type of instance-based learning, a method of supervised machine learning. It is used for classification and regression tasks.

In k-NN, the algorithm is given a labeled training dataset and a set of test data. To make a prediction for a test instance, the algorithm looks at the k nearest neighbors in the training dataset, based on the distance between the test instance and the training instances. The prediction is then made based on the majority class among the k nearest neighbors. For classification tasks, the prediction is the class with the most neighbors. For regression tasks, the prediction is the mean or median of the values of the k nearest neighbors.

Importing the libraries

[1]:
```
import pandas as pd
```

Importing the dataset

[2]:
```
df = pd.read_csv('./datasets/Social_Network_Ads.csv')
df.head()
```

[2]:

	Age	EstimatedSalary	Purchased
0	19	19000	0
1	35	20000	0
2	26	43000	0
3	27	57000	0
4	19	76000	0

[3]:
```
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values
```

Splitting the dataset into the Training set and Test set

[4]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
↪random_state=123)
```

Feature Scaling

k-NN is sensitive to the scale of the features, and it may not perform well if the features have very different scales.

[5]:
```
from sklearn.preprocessing import StandardScaler
```

In order to avoid *information leakage*, it is highly important to keep in mind that only the `transform` method has to be applied on the `X_test`. (μ, σ) are of `X_train` set.

[6]:
```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Training the k-NN model on the training set

[7]:
```
from sklearn.neighbors import KNeighborsClassifier
```

[8]:
```
clf = KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2)
```

[9]:
```
clf.fit(X_train, y_train)
```

[9]:
```
KNeighborsClassifier()
```

Predicting a new result

[10]:
```
clf.predict(sc.transform([[30,87000]]))
```

[10]:
```
array([0])
```

Predicting the test set results

[11]:
```
y_pred = clf.predict(X_test)
```

Displaying the Confusion Matrix

[12]:
```
from sklearn.metrics import confusion_matrix
```

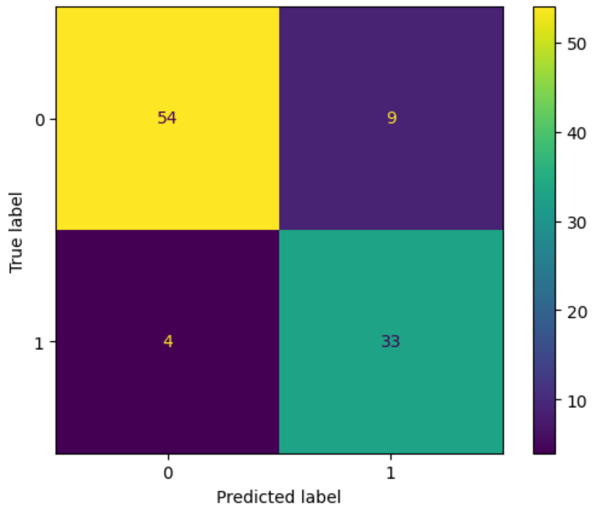
[13]:
```
cm = confusion_matrix(y_test, y_pred, labels=clf.classes_)
```

[14]:
```
cm
```

[14]:
```
array([[54,  9],
       [ 4, 33]])
```

[15]:
```
from sklearn.metrics import ConfusionMatrixDisplay
```

[16]:
```
ConfusionMatrixDisplay(cm, display_labels=clf.classes_).plot();
```



[17]:
```
from sklearn.metrics import accuracy_score
```

[18]:
```
print(f'Accuracy = {accuracy_score(y_test, y_pred):.2f}')
```

```
Accuracy = 0.87
```

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.crosstab.html

[19]:
```
pd.crosstab(y_test, y_pred, rownames=['Expected'], colnames=['Predicted'],
↪margins=True)
```

[19]:
```
Predicted   0   1  All
Expected
0          54   9   63
1           4  33   37
All        58  42  100
```

[20]:
```python
from sklearn.metrics import classification_report
```

[21]:
```python
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.93      0.86      0.89        63
           1       0.79      0.89      0.84        37

    accuracy                           0.87       100
   macro avg       0.86      0.87      0.86       100
weighted avg       0.88      0.87      0.87       100
```

k-NN is a simple and effective method for classification and regression tasks, and it is easy to understand and implement. However, it can be computationally expensive to find the *k* nearest neighbors for each test instance, especially for large datasets.

4 K-Means for Clustering

Student's name	....	....	....
Score **/20**	....	....	....

Detailed Credits

Anticipation ***(4 points)***	....	....	....
Management ***(2 points)***	....	....	....
Testing ***(7 points)***	....	....	....
Data Logging ***(3 points)***	....	....	....
Interpretation ***(4 points)***	....	....	....

Motivations

- ★ K-means clustering is a widely used method for partitioning a dataset into a set of clusters, where each cluster consists of data points that are similar to each other. This can be useful for a variety of applications, including data compression, anomaly detection, and customer segmentation.
- ★ It can also help to identify outliers and anomalies in the data, which can be useful for identifying errors or identifying new opportunities for analysis.

The notebook is available at `https://github.com/a-mhamdi/mlpy/` ⟶ *Codes* ⟶ *Python* ⟶ *k-means-clustering.ipynb*

In unsupervised learning, the algorithm is given a dataset and is asked to learn the underlying structure of the data. The goal is to find patterns or relationships in the data that can be used to group the data points into clusters or to reduce the dimensionality of the data.

Some examples of unsupervised learning algorithms include:

- *K*-Means clustering;
- Principal Component Analysis (PCA); and
- Autoencoders.

These algorithms can be used for tasks such as image compression, anomaly detection, and customer segmentation.

K-Means clustering is a method of unsupervised machine learning used to partition a dataset into k clusters, where k is a user-specified number. The goal of *K*-Means clustering is to minimize the sum of squared distances between the points in each cluster and its centroid.

Importing the libraries

[1]:
```
import pandas as pd
import matplotlib.pyplot as plt
```

[2]:
```
# Show plots in an interactive format, e.g., zooming, saving, etc
%matplotlib inline
```

[3]:
```
plt.style.use('ggplot')
```

Importing the dataset

[4]:
```
df = pd.read_csv('./datasets/Mall_Customers.csv')
```

[5]:
```
df.head()
```

[5]:

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

[6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   CustomerID              200 non-null    int64
 1   Gender                  200 non-null    object
 2   Age                     200 non-null    int64
 3   Annual Income (k$)      200 non-null    int64
 4   Spending Score (1-100)  200 non-null    int64
dtypes: int64(4), object(1)
memory usage: 7.9+ KB
```

[7]:
```
df.describe()
```

[7]:

	CustomerID	Age	Annual Income (k$)	Spending Score (1-100)
count	200.000000	200.000000	200.000000	200.000000
mean	100.500000	38.850000	60.560000	50.200000
std	57.879185	13.969007	26.264721	25.823522
min	1.000000	18.000000	15.000000	1.000000
25%	50.750000	28.750000	41.500000	34.750000
50%	100.500000	36.000000	61.500000	50.000000
75%	150.250000	49.000000	78.000000	73.000000

```
max     200.000000    70.000000            137.000000                99.000000
```

[8]:
```
df.rename(columns={'Annual Income (k$)':'Annual Income', 'Spending Score␣
↪(1-100)': 'Spending Score'}, inplace=True)
```

[9]:
```
X = df.drop(columns=['CustomerID', 'Age', 'Gender']).values
X[:10, :]
```

[9]:
```
array([[15, 39],
       [15, 81],
       [16,  6],
       [16, 77],
       [17, 40],
       [17, 76],
       [18,  6],
       [18, 94],
       [19,  3],
       [19, 72]])
```

Import *K*–Means class

[10]:
```
from sklearn.cluster import KMeans
```

Training the *K*-Means model on the dataset

This code will create a *K*-Means model with 5 clusters and fit it to the data. It will then make predictions about which cluster each data point belongs to.

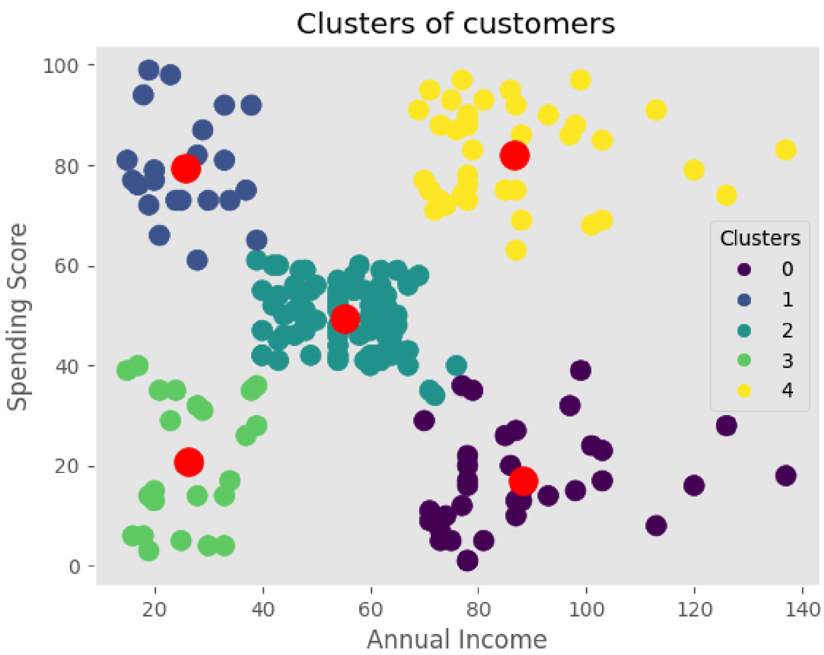
[12]:
```
kmeans = KMeans(n_clusters=5, init='k-means++', n_init='auto')
y_pred = kmeans.fit_predict(X)
```

[13]:
```
centers = kmeans.cluster_centers_
centers
```

[13]:
```
array([[88.2       , 17.11428571],
       [25.72727273, 79.36363636],
       [55.2962963 , 49.51851852],
       [26.30434783, 20.91304348],
       [86.53846154, 82.12820513]])
```

Visualizing the clusters

[14]:
```
fig, ax = plt.subplots()
scatter =  ax.scatter(X[:, 0], X[:, 1], c=y_pred, s=100)
legend = ax.legend(*scatter.legend_elements(), title='Clusters')
ax.add_artist(legend)
ax.scatter(centers[:, 0], centers[:, 1], c='red', s=200)
ax.set_title('Clusters of customers')
ax.set_xlabel('Annual Income')
ax.set_ylabel('Spending Score')
ax.grid()
```



Unsupervised learning can be useful when there is no labeled training data available, or when the goal is to discover patterns or relationships in the data rather than to make predictions. However, it can be more difficult to evaluate the performance of unsupervised learning algorithms, as there is no ground truth to compare the predictions to.

K-Means clustering is a fast and efficient method for clustering large datasets, and is often used as a baseline method for comparison with other clustering algorithms. However, it can be sensitive to the initial selection of centroids, and may not always find the optimal clusters if the data is not well-separated or has a non-convex shape. It is also limited to spherical clusters and may not work well for clusters with more complex shapes.

5 | Binary Classifier using ANN

Student's name	....	....	....
Score /20	....	....	....
Detailed Credits			
Anticipation *(4 points)*	....	....	....
Management *(2 points)*	....	....	....
Testing *(7 points)*	....	....	....
Data Logging *(3 points)*	....	....	....
Interpretation *(4 points)*	....	....	....

Motivations

★ Artificial neural networks (ANNs) are a powerful tool for binary classification tasks, which involve predicting a binary outcome (e.g., "yes" or "no") based on input data. ANNs are able to learn complex relationships between the input data and the output labels, which makes them well-suited for tasks with a large number of features or a complex underlying structure.

★ ANNs are highly flexible and can be trained on a wide range of data types, including continuous and categorical variables. They can also handle missing values and handle large amounts of data efficiently. This makes them a good choice for tasks where the data is noisy or high-dimensional.

The notebook is available at `https://github.com/a-mhamdi/mlpy/` ⟶ *Codes* ⟶ *Python* ⟶ *artificial-neural-network.ipynb*

Artificial neural networks (ANN) are commonly used for classification tasks because they are able to learn complex relationships between the input features and the target class. They are particularly useful when the relationship is non-linear, as they are able to learn and model the inputs-outputs mapping using multiple hidden layers of interconnected neurons.

ANN are also able to handle large amounts of data and can learn from it without being explicitly programmed with a set of rules or a decision tree. This allows them to be very flexible and adaptable, and makes them well-suited for tasks that are difficult to define using traditional programming techniques.

There are several advantages to using neural networks for classification tasks:

1. They are able to learn complex relationships between the input features and the target class;
2. They are able to handle large amounts of data;

3. They can learn from unstructured data;
4. They are flexible and adaptable;
5. They can be trained to perform well on a wide range of classification tasks.

Importing the libraries

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

[2]:
```
np.set_printoptions(precision=2)
```

Importing the dataset

[3]:
```
df = pd.read_csv("./datasets/Churn_Modelling.csv")
```

[4]:
```
df = df.dropna(how="any", axis=0)
```

[5]:
```
df.head()
```

[5]:
```
   RowNumber  CustomerId   Surname  CreditScore Geography  Gender  Age  \
0          1    15634602  Hargrave          619    France  Female   42
1          2    15647311      Hill          608     Spain  Female   41
2          3    15619304      Onio          502    France  Female   42
3          4    15701354      Boni          699    France  Female   39
4          5    15737888  Mitchell          850     Spain  Female   43

   Tenure    Balance  NumOfProducts  HasCrCard  IsActiveMember  \
0       2       0.00              1          1               1
1       1   83807.86              1          0               1
2       8  159660.80              3          1               0
3       1       0.00              2          0               0
4       2  125510.82              1          1               1

   EstimatedSalary  Exited
0        101348.88       1
1        112542.58       0
2        113931.57       1
3         93826.63       0
4         79084.10       0
```

[6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   RowNumber        10000 non-null  int64
 1   CustomerId       10000 non-null  int64
 2   Surname          10000 non-null  object
```

```
 3   CreditScore      10000 non-null  int64
 4   Geography        10000 non-null  object
 5   Gender           10000 non-null  object
 6   Age              10000 non-null  int64
 7   Tenure           10000 non-null  int64
 8   Balance          10000 non-null  float64
 9   NumOfProducts    10000 non-null  int64
 10  HasCrCard        10000 non-null  int64
 11  IsActiveMember   10000 non-null  int64
 12  EstimatedSalary  10000 non-null  float64
 13  Exited           10000 non-null  int64
dtypes: float64(2), int64(9), object(3)
memory usage: 1.1+ MB
```

[7]:
```
df.describe()
```

[7]:
```
         RowNumber    CustomerId   CreditScore           Age        Tenure  \
count  10000.00000  1.000000e+04  10000.000000  10000.000000  10000.000000
mean    5000.50000  1.569094e+07    650.528800     38.921800      5.012800
std     2886.89568  7.193619e+04     96.653299     10.487806      2.892174
min        1.00000  1.556570e+07    350.000000     18.000000      0.000000
25%     2500.75000  1.562853e+07    584.000000     32.000000      3.000000
50%     5000.50000  1.569074e+07    652.000000     37.000000      5.000000
75%     7500.25000  1.575323e+07    718.000000     44.000000      7.000000
max    10000.00000  1.581569e+07    850.000000     92.000000     10.000000

             Balance  NumOfProducts    HasCrCard  IsActiveMember  \
count   10000.000000   10000.000000  10000.00000    10000.000000
mean    76485.889288       1.530200      0.70550        0.515100
std     62397.405202       0.581654      0.45584        0.499797
min         0.000000       1.000000      0.00000        0.000000
25%         0.000000       1.000000      0.00000        0.000000
50%     97198.540000       1.000000      1.00000        1.000000
75%    127644.240000       2.000000      1.00000        1.000000
max    250898.090000       4.000000      1.00000        1.000000

       EstimatedSalary        Exited
count     10000.000000  10000.000000
mean     100090.239881      0.203700
std       57510.492818      0.402769
min          11.580000      0.000000
25%       51002.110000      0.000000
50%      100193.915000      0.000000
75%      149388.247500      0.000000
max      199992.480000      1.000000
```

[8]:
```
X = df.iloc[:, 3:-1].values
y = df.iloc[:, -1].values
```

Data preprocessing

[9]:
```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
```

[10]:
```
le = LabelEncoder()
ohe = OneHotEncoder()
```

[11]:
```
X[:, 2] = le.fit_transform(X[:, 2])
```

[12]:
```
from sklearn.compose import ColumnTransformer
```

[13]:
```
ct = ColumnTransformer([("ohe", ohe, [1])], remainder='passthrough')
X = np.array(ct.fit_transform(X))
```

[14]:
```
X[:5, :]
```

[14]:
```
array([[1.0, 0.0, 0.0, 619, 0, 42, 2, 0.0, 1, 1, 1, 101348.88],
       [0.0, 0.0, 1.0, 608, 0, 41, 1, 83807.86, 1, 0, 1, 112542.58],
       [1.0, 0.0, 0.0, 502, 0, 42, 8, 159660.8, 3, 1, 0, 113931.57],
       [1.0, 0.0, 0.0, 699, 0, 39, 1, 0.0, 2, 0, 0, 93826.63],
       [0.0, 0.0, 1.0, 850, 0, 43, 2, 125510.82, 1, 1, 1, 79084.1]],
      dtype=object)
```

[15]:
```
X = np.asarray(X, dtype=np.float64)
```

[16]:
```
X[:5, :]
```

[16]:
```
array([[1.00e+00, 0.00e+00, 0.00e+00, 6.19e+02, 0.00e+00, 4.20e+01,
        2.00e+00, 0.00e+00, 1.00e+00, 1.00e+00, 1.00e+00, 1.01e+05],
       [0.00e+00, 0.00e+00, 1.00e+00, 6.08e+02, 0.00e+00, 4.10e+01,
        1.00e+00, 8.38e+04, 1.00e+00, 0.00e+00, 1.00e+00, 1.13e+05],
       [1.00e+00, 0.00e+00, 0.00e+00, 5.02e+02, 0.00e+00, 4.20e+01,
        8.00e+00, 1.60e+05, 3.00e+00, 1.00e+00, 0.00e+00, 1.14e+05],
       [1.00e+00, 0.00e+00, 0.00e+00, 6.99e+02, 0.00e+00, 3.90e+01,
        1.00e+00, 0.00e+00, 2.00e+00, 0.00e+00, 0.00e+00, 9.38e+04],
       [0.00e+00, 0.00e+00, 1.00e+00, 8.50e+02, 0.00e+00, 4.30e+01,
        2.00e+00, 1.26e+05, 1.00e+00, 1.00e+00, 1.00e+00, 7.91e+04]])
```

[17]:
```
from sklearn.model_selection import train_test_split
```

[18]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=.8,␣
↪random_state=123)
```

[19]:
```
from sklearn.preprocessing import MinMaxScaler
```

[20]:
```
sc = MinMaxScaler()
```

[21]:
```
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

[22]:
```
print(X_train[:5, :])
```

```
[[1.   0.   0.   0.75 1.   0.16 1.   0.   0.33 1.   1.   0.27]
 [1.   0.   0.   0.51 0.   0.28 1.   0.   0.67 1.   0.   0.66]
 [0.   0.   1.   0.87 1.   0.23 0.3  0.   0.33 0.   0.   0.41]
 [1.   0.   0.   0.69 1.   0.3  0.9  0.   0.33 1.   0.   0.2 ]
 [1.   0.   0.   0.71 1.   0.2  0.3  0.58 0.33 1.   0.   0.57]]
```

Build the classifier `clf`

[23]:
```
from keras.models import Sequential
from keras.layers import Dense
```

```
2023-11-30 01:23:04.966677: I tensorflow/core/platform/cpu_feature_guard.cc:
↪193]
This TensorFlow binary is optimized with oneAPI Deep Neural Network Library
(oneDNN) to use the following CPU instructions in performance-critical
operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
2023-11-30 01:23:05.536785: W
tensorflow/compiler/xla/stream_executor/platform/default/dso_loader.cc:64]␣
↪Could
not load dynamic library 'libcudart.so.11.0'; dlerror: libcudart.so.11.0:␣
↪cannot
open shared object file: No such file or directory
2023-11-30 01:23:05.536804: I
tensorflow/compiler/xla/stream_executor/cuda/cudart_stub.cc:29] Ignore above
cudart dlerror if you do not have a GPU set up on your machine.
2023-11-30 01:23:06.697552: W
tensorflow/compiler/xla/stream_executor/platform/default/dso_loader.cc:64]␣
↪Could
not load dynamic library 'libnvinfer.so.7'; dlerror: libnvinfer.so.7: cannot
open shared object file: No such file or directory
2023-11-30 01:23:06.697725: W
tensorflow/compiler/xla/stream_executor/platform/default/dso_loader.cc:64]␣
↪Could
not load dynamic library 'libnvinfer_plugin.so.7'; dlerror:
libnvinfer_plugin.so.7: cannot open shared object file: No such file or
directory
2023-11-30 01:23:06.697739: W
tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Cannot
dlopen some TensorRT libraries. If you would like to use Nvidia GPU with
TensorRT, please make sure the missing libraries mentioned above are installed
properly.
```

[24]:
```
clf = Sequential()
ndim = X_train.shape[1]
clf.add(Dense(units=8, activation="relu", input_dim=ndim))
clf.add(Dense(units=4, activation="relu"))
clf.add(Dense(units=4, activation="relu"))
clf.add(Dense(units=1, activation="sigmoid"))
```

```
2023-11-30 01:23:07.562116: W
tensorflow/compiler/xla/stream_executor/platform/default/dso_loader.cc:64]␣
↪Could
not load dynamic library 'libcuda.so.1'; dlerror: libcuda.so.1: cannot open
shared object file: No such file or directory
2023-11-30 01:23:07.562177: W
tensorflow/compiler/xla/stream_executor/cuda/cuda_driver.cc:265] failed call␣
↪to
cuInit: UNKNOWN ERROR (303)
2023-11-30 01:23:07.562196: I
tensorflow/compiler/xla/stream_executor/cuda/cuda_diagnostics.cc:156] kernel
driver does not appear to be running on this host (e590):
/proc/driver/nvidia/version does not exist
2023-11-30 01:23:07.562409: I tensorflow/core/platform/cpu_feature_guard.cc:
↪193]
This TensorFlow binary is optimized with oneAPI Deep Neural Network Library
(oneDNN) to use the following CPU instructions in performance-critical
operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
```

Insights about `clf`

[25]:
```
clf.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 8)                 104

 dense_1 (Dense)             (None, 4)                 36

 dense_2 (Dense)             (None, 4)                 20

 dense_3 (Dense)             (None, 1)                 5

=================================================================
Total params: 165
Trainable params: 165
Non-trainable params: 0
_________________________________________________________________
```

Compile `clf`

[26]:
```
clf.compile(optimizer="adam",
            loss="binary_crossentropy",
            metrics=['Accuracy', 'Precision', 'Recall'])
```

Train and evaluate clf

[27]:
```
clf.fit(X_train, y_train, batch_size=16, epochs=32);
```

```
Epoch 1/32
500/500 [==============================] - 1s 1ms/step - loss: 0.5106 -
Accuracy: 0.7904 - precision: 0.3533 - recall: 0.0400
Epoch 2/32
500/500 [==============================] - 1s 1ms/step - loss: 0.4553 -
Accuracy: 0.7971 - precision: 0.0000e+00 - recall: 0.0000e+00
Epoch 3/32
500/500 [==============================] - 1s 1ms/step - loss: 0.4392 -
Accuracy: 0.7971 - precision: 0.0000e+00 - recall: 0.0000e+00
Epoch 4/32
500/500 [==============================] - 1s 1ms/step - loss: 0.4218 -
Accuracy: 0.7983 - precision: 0.8000 - recall: 0.0074
Epoch 5/32
500/500 [==============================] - 1s 1ms/step - loss: 0.4056 -
Accuracy: 0.8145 - precision: 0.6980 - recall: 0.1510
Epoch 6/32
500/500 [==============================] - 1s 1ms/step - loss: 0.3941 -
Accuracy: 0.8254 - precision: 0.6733 - recall: 0.2705
Epoch 7/32
500/500 [==============================] - 1s 1ms/step - loss: 0.3883 -
Accuracy: 0.8319 - precision: 0.6853 - recall: 0.3167
Epoch 8/32
500/500 [==============================] - 1s 1ms/step - loss: 0.3840 -
Accuracy: 0.8364 - precision: 0.6817 - recall: 0.3629
Epoch 9/32
500/500 [==============================] - 1s 1ms/step - loss: 0.3799 -
Accuracy: 0.8364 - precision: 0.6718 - recall: 0.3783
Epoch 10/32
500/500 [==============================] - 1s 1ms/step - loss: 0.3779 -
Accuracy: 0.8400 - precision: 0.6850 - recall: 0.3913
Epoch 11/32
500/500 [==============================] - 1s 1ms/step - loss: 0.3753 -
Accuracy: 0.8440 - precision: 0.7045 - recall: 0.3980
Epoch 12/32
500/500 [==============================] - 1s 1ms/step - loss: 0.3728 -
Accuracy: 0.8465 - precision: 0.7112 - recall: 0.4097
Epoch 13/32
500/500 [==============================] - 1s 1ms/step - loss: 0.3711 -
Accuracy: 0.8470 - precision: 0.7157 - recall: 0.4079
Epoch 14/32
500/500 [==============================] - 1s 1ms/step - loss: 0.3700 -
Accuracy: 0.8489 - precision: 0.7216 - recall: 0.4153
Epoch 15/32
500/500 [==============================] - 1s 1ms/step - loss: 0.3688 -
Accuracy: 0.8497 - precision: 0.7223 - recall: 0.4214
Epoch 16/32
500/500 [==============================] - 1s 1ms/step - loss: 0.3673 -
```

```
Accuracy: 0.8499 - precision: 0.7279 - recall: 0.4153
Epoch 17/32
500/500 [==============================] - 1s 1ms/step - loss: 0.3673 -
Accuracy: 0.8503 - precision: 0.7216 - recall: 0.4264
Epoch 18/32
500/500 [==============================] - 1s 1ms/step - loss: 0.3653 -
Accuracy: 0.8528 - precision: 0.7350 - recall: 0.4288
Epoch 19/32
500/500 [==============================] - 1s 1ms/step - loss: 0.3650 -
Accuracy: 0.8524 - precision: 0.7361 - recall: 0.4245
Epoch 20/32
500/500 [==============================] - 1s 1ms/step - loss: 0.3646 -
Accuracy: 0.8529 - precision: 0.7333 - recall: 0.4319
Epoch 21/32
500/500 [==============================] - 1s 1ms/step - loss: 0.3629 -
Accuracy: 0.8539 - precision: 0.7441 - recall: 0.4264
Epoch 22/32
500/500 [==============================] - 1s 1ms/step - loss: 0.3614 -
Accuracy: 0.8531 - precision: 0.7419 - recall: 0.4233
Epoch 23/32
500/500 [==============================] - 1s 1ms/step - loss: 0.3614 -
Accuracy: 0.8551 - precision: 0.7432 - recall: 0.4368
Epoch 24/32
500/500 [==============================] - 1s 1ms/step - loss: 0.3603 -
Accuracy: 0.8547 - precision: 0.7444 - recall: 0.4325
Epoch 25/32
500/500 [==============================] - 1s 1ms/step - loss: 0.3589 -
Accuracy: 0.8533 - precision: 0.7341 - recall: 0.4338
Epoch 26/32
500/500 [==============================] - 1s 1ms/step - loss: 0.3578 -
Accuracy: 0.8545 - precision: 0.7393 - recall: 0.4368
Epoch 27/32
500/500 [==============================] - 1s 1ms/step - loss: 0.3568 -
Accuracy: 0.8549 - precision: 0.7416 - recall: 0.4368
Epoch 28/32
500/500 [==============================] - 1s 1ms/step - loss: 0.3575 -
Accuracy: 0.8561 - precision: 0.7453 - recall: 0.4418
Epoch 29/32
500/500 [==============================] - 1s 1ms/step - loss: 0.3562 -
Accuracy: 0.8558 - precision: 0.7503 - recall: 0.4331
Epoch 30/32
500/500 [==============================] - 1s 1ms/step - loss: 0.3549 -
Accuracy: 0.8546 - precision: 0.7478 - recall: 0.4276
Epoch 31/32
500/500 [==============================] - 1s 1ms/step - loss: 0.3548 -
Accuracy: 0.8550 - precision: 0.7360 - recall: 0.4449
Epoch 32/32
500/500 [==============================] - 1s 1ms/step - loss: 0.3545 -
Accuracy: 0.8558 - precision: 0.7492 - recall: 0.4344
```

[28]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

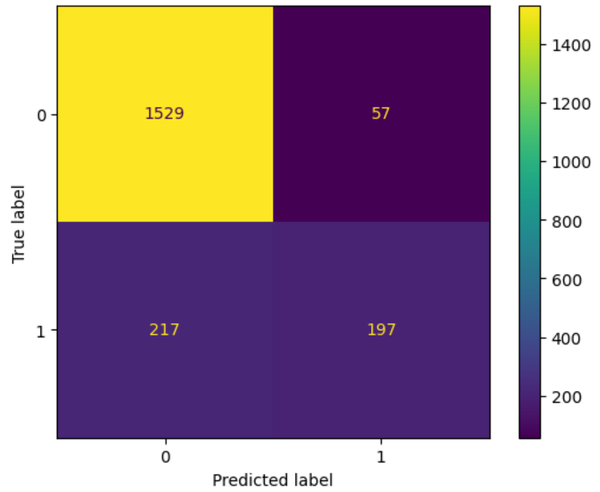
[29]:
```
y_pred = clf.predict(X_test)
y_pred = ( y_pred > .5 ).astype(int)
```

```
63/63 [==============================] - 0s 773us/step
```

Print the confusion matrix

[30]:
```
cm = confusion_matrix(y_test, y_pred)
```

[31]:
```
ConfusionMatrixDisplay(cm).plot();
```



[32]:
```
print(y_test.shape)
```

```
(2000,)
```

[33]:
```
print(y_pred.shape)
y_pred = y_pred.reshape(len(y_pred),)
print(y_pred.shape)
```

```
(2000, 1)
(2000,)
```

[34]:
```
pd.crosstab(y_test, y_pred, rownames=["Expected"], colnames=["Predicted"],␣
↪margins=True)
```

[34]:

Predicted	0	1	All
Expected			
0	1529	57	1586
1	217	197	414
All	1746	254	2000

[35]:
```
y_test = y_test.reshape(len(y_test), 1)
y_pred = y_pred.reshape(len(y_pred), 1)
print(np.concatenate((y_test[:10], y_pred[:10]), axis=1))
```

```
[[0 0]
 [0 0]
 [0 0]
```

```
 [0 0]
 [0 0]
 [0 0]
 [1 0]
 [1 1]
 [0 0]
 [0 0]]
```

[36]:
```python
from sklearn.metrics import classification_report
```

[37]:
```python
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.96      0.92      1586
           1       0.78      0.48      0.59       414

    accuracy                           0.86      2000
   macro avg       0.83      0.72      0.75      2000
weighted avg       0.85      0.86      0.85      2000
```

It is important to note that neural networks can be more computationally intensive to train and may require more data and more time to achieve good performance, compared to some other classification algorithms. Additionally, they can be more difficult to interpret and understand, as they learn patterns in the data through the weights and biases of the network rather than through explicit rules.

The overall scope of this manual is to introduce **Machine Learning**, through some numeric simulations, to the students at the department of **Electrical Engineering**.

The topics discussed in this manuscript are as follow:

① Getting started with *Python*

② Linear Regression

③ Classification

④ Clustering

⑤ ANN

Python; *Jupyter*; *NumPy*; *Matplotlib*; *scikit-learn*; machine learning; linear regression; classification; clustering; deep learning.