International Journal of Education, Modern Management, Applied Science & Social Science (IJEMMASSS)
ISSN : 2581-9925, Impact Factor: 6.882, Volume 06, No. 01(IV), January - March, 2024, pp. 75-79

MACHINE LEARNING IN PYTHON

Nirav K. Dave[*]
Bhavesh L. Dhandhukiya[**]
Sagar B Vala[***]

ABSTRACT

The goal of machine learning is to derive knowledge from data. Also referred to as statistical learning or predictive analytics, this field of study sits at the nexus of computer science, artificial intelligence, and statistics. In recent years, machine learning techniques have become widely used in daily life.

Keywords: *Machine Learning, Numpy, Scikit, Jupyter Notebook.*

Introduction

In the past, sophisticated programs were unable to gather vast amounts of data in an organized way, necessitating a great deal of work to organize them. In addition, even "intelligent" apps relied heavily on manually designed "if" and "else" rules to process data or respond to user input. There is a chance that some of the data in this case will damage the systems. However, there are two significant drawbacks to making decisions using hand-coded rules:

- The logic required to decide is specific to a single domain and task. Changing the task even slightly might require a rewrite of the whole system.
- Designing rules requires a deep understanding of how a decision should be made by a human expert.

However, with machine learning, an algorithm may decide the systematic collecting and arrangement of data with profitable results just by giving a program with a vast collection of data.

Problems Machine Learning can Solve

Machine learning algorithms that automate decision-making by extrapolating from well-known examples have shown to be the most effective. The user gives the algorithm pairs of inputs and intended outputs in this context, which is referred to as supervised learning, and the algorithm figures out how to produce the desired output given an input. Additionally, A well-trained database is the foundation for intelligent output from machine learning.

Why Python

The majority of programming languages can be used to produce output in machine learning techniques, but Python is currently the most well-known language for handling machine learning problems. This is partly because Python includes sizable modules that are simple for developers to implement. Furthermore, anyone can freely implement the Python framework or modules because Python is an open source language. Python may also readily interpret machine complexity using this method.

[*] Assistant Professor, Shree Swaminarayan College of Computer Science, Bhavnagar, India.
[**] Assistant Professor, Shree Swaminarayan College of Computer Science, Bhavnagar, India.
[***] Assistant Professor, Shree Swaminarayan College of Computer Science, Bhavnagar, India.

Use of Scikit-Learn in Machine Learning

Since scikit-learn is an open source project, anyone may readily acquire the source code to observe what's happening behind the scenes and use it for free. The scikit-learn project has a vibrant user community and is always being developed and enhanced. It includes extensive documentation for every algorithm in addition to some cutting-edge machine learning algorithms. It is the most well-known Python machine learning package and a very popular tool. Let's examine the Python implementation.

Installing Scikit-Learn

NumPy and SciPy are two additional Python packages that scikit-learn depends on. You need also install IPython, the Jupyter Notebook, and matplotlib for plotting and interactive programming. One of the prepackaged Python distributions listed below is what we advise using as it has all the required packages. A free edition of Python designed for scientific computing, optimized for Windows. NumPy, SciPy, matplotlib, pandas, IPython, and scikit-learn are all included with Python(x,y). All of these packages can be installed with pip if you already have a Python installation configured:

```
$ pip install numpy scipy matplotlib ipython scikit-learn pandas
```

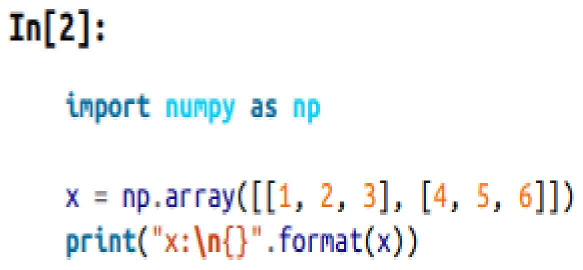
Essential Libraries and Tools

- Although you should be familiar with scikit-learn, there are a few more packages that can help you get better results. SciPy and NumPy are scientific Python modules that form the basis of scikit-learn. In addition to NumPy and SciPy, we will also use pandas and matplotlib. We will also introduce you to Jupyter Notebook, an interactive browser-based programming environment. To put it briefly, the following details about these resources will assist you in getting the most out of scikit-learn. **Jupyter Notebook**

Code can be run interactively in a browser using the Jupyter Notebook. Data scientists use it extensively because it is an excellent tool for exploratory data analysis. While there are other programming languages supported by the Jupyter Notebook, we just require support for Python. Including code, text, and photos is simple using the Jupyter Notebook—in fact, this entire book was written in one. You can download each of the code samples we provide from GitHub.

- **NumPy**

One of the core Python packages for scientific computing is NumPy. It has features for pseudorandom number generators, multidimensional arrays, and advanced mathematical operations like the Fourier transform and linear algebra procedures. The NumPy array is the basic data structure of scikit-learn. NumPy arrays are used as the input format for scikit-learn. You will need to convert any existing data to a NumPy array. The multidimensional (n-dimensional) array of the nd-array class is the central feature of NumPy. The array's items must all be of the same type. This is how a NumPy array appears:



```
In[2]:

    import numpy as np

    x = np.array([[1, 2, 3], [4, 5, 6]])
    print("x:\n{}".format(x))
```

Out[2]:

x: [[1 2 3] [4 5 6]]

- **SciPy**

Python routines for scientific computation are gathered under the SciPy package. Advanced linear algebra procedures, mathematical function optimization, signal processing, specific mathematical functions, and statistical distributions are only a few of the features it offers. SciPy's function library is utilized by scikit-learn to implement its algorithms. For us, SciPy is the most crucial component of SciPy. sparse: this yields sparse matrices, another type of data format utilized by scikit-learn. If we wish to store a 2D array with primarily zero entries, we use sparse matrices:

In[3]:

```
from scipy import sparse

# Create a 2D NumPy array with a diagonal of ones, and zeros everywhere else
eye = np.eye(4)
print("NumPy array:\n{}".format(eye))
```

Out[3]:

```
NumPy array:
[[ 1.  0.  0.  0.]
 [ 0.  1.  0.  0.]
 [ 0.  0.  1.  0.]
 [ 0.  0.  0.  1.]]
```

In[4]:

```
# Convert the NumPy array to a SciPy sparse matrix in CSR format
# Only the nonzero entries are stored
sparse_matrix = sparse.csr_matrix(eye)
print("\nSciPy sparse CSR matrix:\n{}".format(sparse_matrix))
```

Out[4]:

```
SciPy sparse CSR matrix:
  (0, 0)    1.0
  (1, 1)    1.0
  (2, 2)    1.0
  (3, 3)    1.0
```

We must directly generate sparse representations since dense representations of sparse data are typically not able to be created since they would not fit into memory. Here's how to use the COO format to generate the same sparse matrix as before:

In[5]:

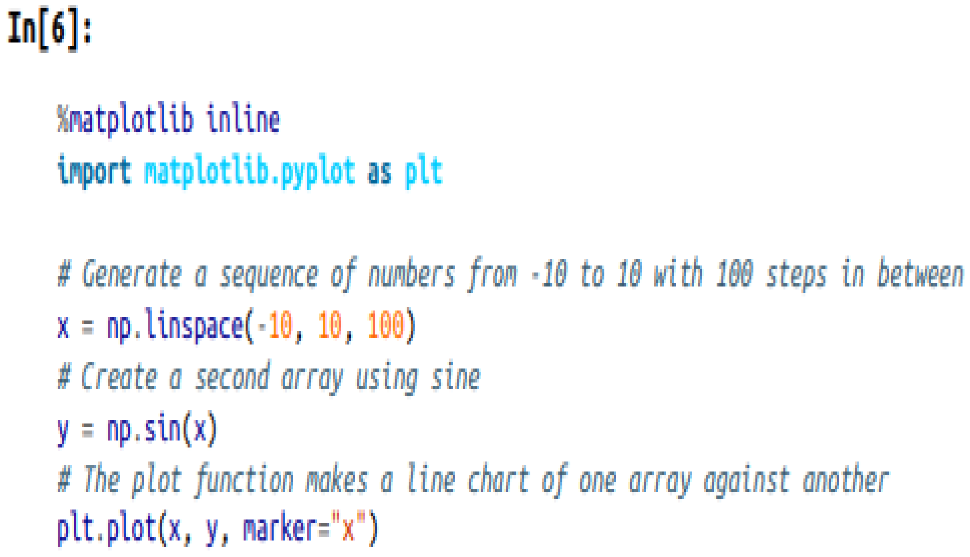
```
data = np.ones(4)
row_indices = np.arange(4)
col_indices = np.arange(4)
eye_coo = sparse.coo_matrix((data, (row_indices, col_indices)))
print("COO representation:\n{}".format(eye_coo))
```

Out[5]:

```
COO representation:
  (0, 0)    1.0
  (1, 1)    1.0
  (2, 2)    1.0
  (3, 3)    1.0
```

- **Matplotlib**

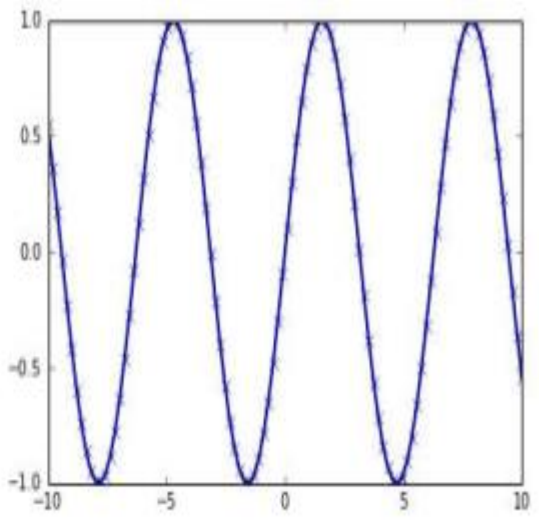
The main Python scientific charting library is called matplotlib. It offers tools for creating line charts, scatter plots, histograms, and other visualizations suitable for publishing. You can gain crucial insights by visualizing your data and various components of your investigation; matplotlib will be the tool of choice for all of our visualizations. You can use the %matplotlib notebook and %matplotlib inline commands to display figures directly in the browser when working within the Jupyter Notebook. This code, for instance, generates the plot shown in Figure 1:



```
In[6]:

%matplotlib inline
import matplotlib.pyplot as plt

# Generate a sequence of numbers from -10 to 10 with 100 steps in between
x = np.linspace(-10, 10, 100)
# Create a second array using sine
y = np.sin(x)
# The plot function makes a line chart of one array against another
plt.plot(x, y, marker="x")
```



- **Pandas**

Pandas is a Python data analysis and wrangling toolkit. The foundation of it is a data structure known as the Data-Frame, which is based on the R Data-Frame. A pandas Data-Frame is, in essence, a table that resembles an Excel spreadsheet. Pandas offers a wide range of operations and modifications for this data, including SQL-like queries and table joins. Whereas NumPy mandates that every element in an array be of the same type, pandas permits different types for each column (such as dates, integers, floating-point numbers, and strings, among others). The capacity of pandas to ingest data from a wide range of file types and databases, including SQL, Excel files, and comma-separated. Here is a small example of creating a Data-Frame using a dictionary:

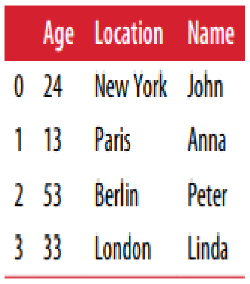
Output

In[7]:

```
import pandas as pd

# create a simple dataset of people
data = {'Name': ["John", "Anna", "Peter", "Linda"],
        'Location' : ["New York", "Paris", "Berlin", "London"],
        'Age' : [24, 13, 53, 33]
       }

data_pandas = pd.DataFrame(data)
# IPython.display allows "pretty printing" of dataframes
# in the Jupyter notebook
display(data_pandas)
```



	Age	Location	Name
0	24	New York	John
1	13	Paris	Anna
2	53	Berlin	Peter
3	33	London	Linda

Versions used in this Paper

We are using the following versions of the previously mentioned libraries in this research paper

In[9]:

```
import sys
print("Python version: {}".format(sys.version))

import pandas as pd
print("pandas version: {}".format(pd.__version__))

import matplotlib
print("matplotlib version: {}".format(matplotlib.__version__))

import numpy as np
print("NumPy version: {}".format(np.__version__))

import scipy as sp
print("SciPy version: {}".format(sp.__version__))

import IPython
print("IPython version: {}".format(IPython.__version__))

import sklearn
print("scikit-learn version: {}".format(sklearn.__version__))
```

References

1. Introduction to Machine Learning with Python A Guide for Data Scientists: Andreas C. Müller and Sarah Guido.
2. Deep Learning with Python: FRANÇOIS CHOLLET.

❖◆❖