NumPy

Introduction: NumPy (Numerical Python) is a fundamental Python library for numerical computing. It provides support for large, multi-dimensional arrays and matrices, as well as a wide range of high-level mathematical functions to operate on these arrays. NumPy is a critical library in the data science, scientific computing, and machine learning ecosystems.

Some of the basic operations we can perform with NumPy

1. **Arrays:** NumPy's primary data structure is the ndarray (n-dimensional array). These arrays can have multiple dimensions and are homogeneous, meaning that all elements are of the same data type. We can create NumPy arrays from Python lists or initialize them with specific values.

   Code:

   ```
   import numpy as np
   # Creating a NumPy array from a Python list
   my_array = np.array([1, 2, 3, 4, 5])
   print("Numpy array:")
   print(my_array)
   ```

   Output:

   ```
   Numpy array:
   [1 2 3 4 5]
   ```

2. **Array Operation:** NumPy provides a wide range of mathematical and logical operations for arrays, making it easy to perform element-wise operations.

   Code:

   ```
   # Element-wise addition
   result1 = my_array + 10
   print("Original array:")
   print(my_array)
   print("Element-wise addition result:")
   print(result1)
   # Element-wise multiplication
   result2 = my_array * 2
   ```

```
print("Element-wise multiplication result:")
print(result2)
# Dot product of two arrays
dot_product = np.dot(result1, result2)
print("Element-wise Dot product result:")
print(dot_product)
```

Output:

```
Original array:
[1 2 3 4 5]
Element-wise addition result:
[11 12 13 14 15]
Element-wise multiplication result:
[ 2  4  6  8 10]
Element-wise Dot product result:
410
```

3. **Array Slicing and Indexing:** We can slice and index NumPy arrays to access specific elements or subarrays.

Code:

```
# Slicing
subarray = my_array[1:4]
print("Original array:")
print(my_array)
print("Sub-array after slicing")
print(subarray)
element = my_array[2]
print("Third element of the array:")
print(element)
```

Output:

```
Original array:
[1 2 3 4 5]
Sub-array after slicing
```

[2 3 4]

Third element of the array:

3

4. **Array Shape and Dimensions:** We can reshape, transpose, and manipulate the dimensions of NumPy arrays.

   Code:

   ```
   # Reshaping
   reshaped_array = my_array.reshape(2, 3)
   print("Originale array:")
   print(my_array)
   print("Reshaped array:")
   print(reshaped_array)
   # Transposing
   transposed_array = np.transpose(reshaped_array)
   print("Transposed array:")
   print(transposed_array)
   ```

   Output:

   ```
   Originale array:
   [1 2 3 4 5 6]
   Reshaped array:
   [[1 2 3]
    [4 5 6]]
   Transposed array:
   [[1 4]
    [2 5]
    [3 6]]
   ```

5. **Array Aggregation and Statistics:** NumPy provides functions for aggregating data in arrays, such as sum, mean, median, min, max, and more.

   Code:

   ```
   # Calculating mean and standard deviation
   mean_value = np.mean(my_array)
   ```

```
std_deviation = np.std(my_array)
print("Mean:")
print(mean_value)
print("Standard deviation:")
print(std_deviation)
```

Output:

```
Mean:
3.5
Standard deviation:
1.707825127659933
```

6. **Random Number Generation:** NumPy includes a random number generation module (np.random) for creating random arrays and samples from various probability distributions.

   Code:

```
# Generating random numbers
random_array = np.random.rand(5)  # Random values between 0 and 1
print("Random array")
print(random_array)
```

   Output:

```
Random array:
[0.34915676 0.22901867 0.24640138 0.13216706 0.0155009 ]
```

7. **Linear Algebra:** NumPy has functions for linear algebra operations like matrix multiplication, determinant calculation, and eigenvalue decomposition.

   Code:

```
# Matrix multiplication
matrix1 = [[1, 2, 3, 4],
           [5, 6, 7, 8],
           [9, 10, 11, 12]]
matrix2 = [[9, 2, 3, 1],
           [10, 5, 7, 8],
           [9, 10, 12, 12],
           [2, 10, 2, 14]]
```

```
product = np.matmul(matrix1, matrix2)
print("Product of two matrix")
print(product)
```

Output:

```
Product of two matrix:
[[ 64  82  61 109]
 [184 190 157 249]
 [304 298 253 389]]
```

8. **File I/O:** NumPy can read and write data to/from files, such as text files and binary files.

   Code:

```
# Saving and loading arrays
np.save('my_array.npy', my_array)
loaded_array = np.load('my_array.npy')
```

Pandas

Introduction: Pandas is a popular open-source Python library that provides powerful data manipulation and analysis tools. It is widely used in data science, machine learning, and data analysis projects. Pandas is built on top of the NumPy library and provides easy-to-use data structures and functions for working with structured data, such as spreadsheets or SQL tables. Some of the fundamental components and operations in Pandas include:

Some of the basic operations we can perform with Pandas

1. **Creating DataFrames and Series:** We can create DataFrames and Series from various data sources.

   Code:

```
import pandas as pd
# Creating a DataFrame from a dictionary
data = {'Name': ['Alice', 'Bob', 'Charlie'],
        'Age': [25, 30, 35]}
df = pd.DataFrame(data)
# Creating a Series from a list
series = pd.Series([10, 20, 30, 40])
print("Data frame:")
print(df)
print("Series:")
print(series)
```

Output:

```
Data frame:
      Name  Age
0    Alice   25
1      Bob   30
2  Charlie   35
Series:
0    10
1    20
2    30
3    40
dtype: int64
```

2. **Viewing Data:**

Code:

```
# Display the first few rows of the DataFrame
print("Displaying  head:")
df.head()
# Get dimensions of the DataFrame
print("Displaying summary of data:")
df.shape
# Display summary information about the DataFrame
print("Displaying summary information of data:")
df.info()
```

Output:

```
Output:
Displaying head:
      Name   Age
0    Alice    25
1    Bob      30
2    Charlie35
Displaying shape of data:
(3, 2)
Displaying summary information of data:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Name    3 non-null      object
 1   Age     3 non-null      int64
dtypes: int64(1), object(1)
memory usage: 176.0+ bytes
```

3. **Indexing and Selection:**

Code:

```
# Selecting a single column
Print("Single column:")
df['Name']
# Slicing rows
Print("Slicing rows:")
df[1:3]
# Conditional selection
Print("Conditional selection:")
df[df['Age'] > 30]
```

Output:

```
Single column:
0      Alice
1        Bob
2    Charlie
Name: Name, dtype: object
Slicing rows:
         Name   Age
1         Bob     30
2         Charlie35
Conditional selection:
Name   Age
2         Charlie35
```

4. **Basic Data Manipulation:**

Code:

```
# Adding a new column
df['City'] = ['New York', 'San Francisco', 'Los Angeles']
print("New data frame after adding new column:")
print(df)
# Renaming columns
df.rename(columns={'Name': 'Full Name', 'Age': 'Years Old'}, inplace=True)
print("New data frame after renaming column:")
print(df)
# Dropping a column
df.drop('City', axis=1, inplace=True)
print("New data frame after dropping column:")
print(df)
# Sorting by a column
df.sort_values(by='Years Old', ascending=False, inplace=True)
print("New data frame after sorting by a column:")
print(df)
```

Output:

```
Original data:
      Name  Age           City
0    Alice   25       New York
1      Bob   30  San Francisco
2  Charlie   35    Los Angeles
New data frame after adding new column:
      Name  Age           City
0    Alice   25       New York
1      Bob   30  San Francisco
2  Charlie   35    Los Angeles
```

New data frame after renaming column:

	Full Name	Years Old	City
0	Alice	25	New York
1	Bob	30	San Francisco
2	Charlie	35	Los Angeles

New data frame after dropping column:

	Full Name	Years Old
0	Alice	25
1	Bob	30
2	Charlie	35

New data frame after sorting by a column:

	Full Name	Years Old
2	Charlie	35
1	Bob	30
0	Alice	250

5. **Aggregation and Summary Statistics:**

Code:

```
# Grouping by a column and calculating mean age
grouped = df.groupby('Years Old')
mean_age = grouped['Years Old'].mean()
# Summary statistics for numeric columns
df.describe()
```

Output:

	Years Old
count	3.0
mean	30.0
std	5.0
min	25.0
25%	27.5
50%	30.0

75% 32.5

max 35.0

6. **Data Cleaning:**

Code:

```
df = pd.read_excel("input.xlsx");
print("Originale data:")
print(df)
# Handling missing values by filling with a specific value
df.fillna(23, inplace=True)
print("Handling missing values by filling with a specific value:")
print(df)
# Handling duplicates
df.drop_duplicates(inplace=True)
print("Removing duplicate value:")
print(df)
# Data type conversions
df['Age'] = df['Age'].astype(float)
print("Data type conversions:")
print(df)
```

Output:

```
Originale data:
     Name   Age       City
0   Sifat  32.0      Dhaka
1   Rakib  22.0  Mymensingh
2  Arafat   NaN    Comilla
3  Shanta  25.0   Netrokona
4  Shisir  33.0      Dhaka
Handling missing values by filling with a specific value:
     Name   Age       City
```

0 Sifat 32.0 Dhaka

1 Rakib 22.0 Mymensingh

2 Arafat 23.0 Comilla

3 Shanta 25.0 Netrokona

4 Shisir 33.0 Dhaka

Removing duplicate value:

	Name	Age	City
0	Sifat	32.0	Dhaka
1	Rakib	22.0	Mymensingh
2	Arafat	23.0	Comilla
3	Shanta	25.0	Netrokona
4	Shisir	33.0	Dhaka

Data type conversions:

	Name	Age	City
0	Sifat	32.0	Dhaka
1	Rakib	22.0	Mymensingh
2	Arafat	23.0	Comilla
3	Shanta	25.0	Netrokona
4	Shisir	33.0	Dhaka

7. **Data Input and Output:**

Code:

```
# Reading data from a xlsx file
df = pd.read_excel('input.xlsx')
print("Data")
print(df)
# Writing data to a CSV file
df.to_csv('output.csv', index=False)
```

Output:

Data:

	Name	Age	City
0	Sifat	32.0	Dhaka
1	Rakib	22.0	Mymensingh
2	Arafat	NaN	Comilla
3	Shanta	25.0	Netrokona
4	Shisir	33.0	Dhaka

Scikit-learn

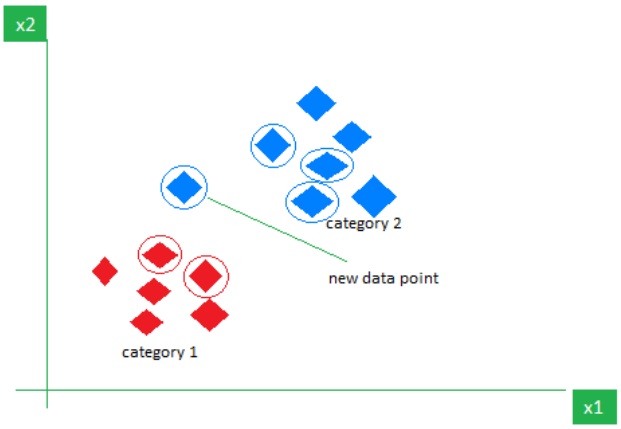
Introduction: scikit-learn is an open-source Python library that implements a range of machine learning, pre-processing, cross-validation, and visualization algorithms using a unified interface. This library supports modern algorithms like KNN, random forest, XGBoost, and SVC. It is constructed over NumPy. Both well-known software companies and the Kaggle competition frequently employ Scikit-learn. It aids in various processes of model building, like model selection, regression, classification, clustering, and dimensionality reduction (parameter selection). Scikit-learn is simple to work with and delivers successful performance. Scikit Learn, though, does not enable parallel processing. We can implement deep learning algorithms in sklearn, though it is not a wise choice, especially if using TensorFlow is an available option.

Important features of scikit-learn:

1. Simple and efficient tools for data mining and data analysis. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means, etc.
2. Accessible to everybody and reusable in various contexts.
3. Built on the top of NumPy, SciPy, and matplotlib.
4. Open source, commercially usable – BSD license.

KNN

Introduction: K-Nearest Neighbours is one of the most basic yet essential classification algorithms in Machine Learning. It belongs to the supervised learning domain and finds intense application in pattern recognition, data mining, and intrusion detection. It is widely disposable in real-life scenarios since it is non-parametric, meaning, it does not make any underlying assumptions about the distribution of data (as opposed to other algorithms such as GMM, which assume a Gaussian distribution of the given data). We are given some prior data (also called training data), which classifies coordinates into groups identified by an attribute. As an example, consider the following table of data points containing two features:



Intuition Behind KNN Algorithm: If we plot these points on a graph, we may be able to locate some clusters or groups. Now, given an unclassified point, we can assign it to a group by observing what group its nearest neighbors belong to. This means a point close to a cluster of points classified as 'Red' has a higher probability of getting classified as 'Red'.

Distance Metrics Used in KNN Algorithm

As we know that the KNN algorithm helps us identify the nearest points or the groups for a query point. But to determine the closest groups or the nearest points for a query point we need some metric. For this purpose, we use below distance metrics:

- Euclidean Distance
- Manhattan Distance
- Minkowski Distance

Euclidean Distance: This is nothing but the cartesian distance between the two points which are in the plane/hyperplane. Euclidean distance can also be visualized as the length of the straight line that joins the two points which are into consideration. This metric helps us calculate the net displacement done between the two states of an object.

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

Manhattan Distance: This distance metric is generally used when we are interested in the total distance traveled by the object instead of the displacement. This metric is calculated by summing the absolute difference between the coordinates of the points in n-dimensions**.**

$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

Minkowski Distance: We can say that the Euclidean, as well as the Manhattan distance, are special cases of the Minkowski distance.

$$d(x, y) = (\sum_{i=1}^{n} (x_i - y_i)^p)^{\frac{1}{p}}$$

Applications of the KNN Algorithm

Data Preprocessing: While dealing with any Machine Learning problem we first perform the EDA part in which if we find that the data contains missing values then there are multiple imputation methods are available as well. One of such method is KNN Imputer which is quite effective ad generally used for sophisticated imputation methodologies.

Pattern Recognition: KNN algorithms work very well if you have trained a KNN algorithm using the MNIST dataset and then performed the evaluation process then you must have come across the fact that the accuracy is too high.

Recommendation Engines: The main task which is performed by a KNN algorithm is to assign a new query point to a pre-existed group that has been created using a huge corpus of datasets. This is exactly what is required in the recommender systems to assign each user to a particular group and then provide them recommendations based on that group's preferences.

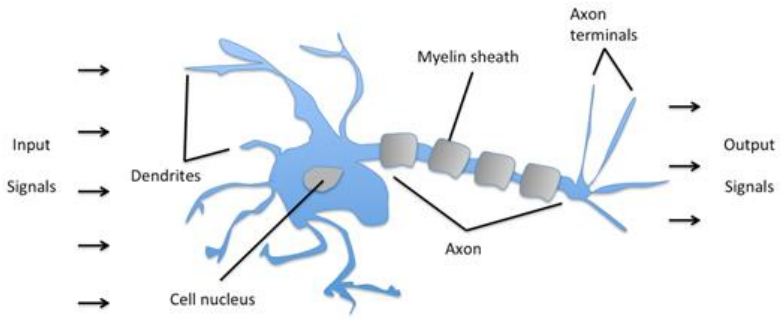
Reference: https://www.geeksforgeeks.org/k-nearest-neighbours/

ANN

Introduction: Artificial neural networks are one of the main tools used in machine learning. As the "neural" part of their name suggests, they are brain-inspired systems that are intended to replicate the way that we humans learn. Neural networks consist of input and output layers, as well as (in most cases) a hidden layer consisting of units that transform the input into something that

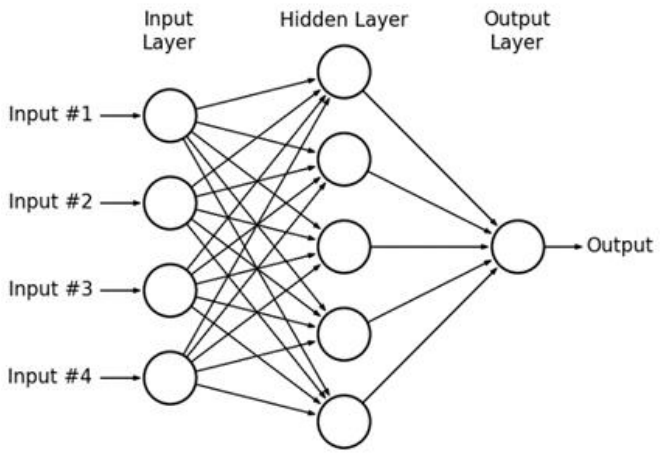
the output layer can use. They are excellent tools for finding patterns that are far too complex or numerous for a human programmer to extract and teach the machine to recognize.

Basic Structure of ANNs: The idea of ANNs is based on the belief that working of the human brain by making the right connections can be imitated using silicon and wires as living neurons and dendrites. The human brain is composed of 86 billion nerve cells called neurons. They are connected to other thousand cells by Axons. Stimuli from the external environment or inputs from sensory organs are accepted by dendrites. These inputs create electric impulses, which quickly travel through the neural network. A neuron can then send the message to another neuron to handle the issue or does not send it forward.



Schematic of a biological neuron.

ANNs are composed of multiple nodes, which imitate biological neurons of the human brain. The neurons are connected by links and they interact with each other. The nodes can take input data and perform simple operations on the data. The result of these operations is passed to other neurons. The output at each node is called its activation or node value. Each link is associated with weight. ANNs are capable of learning, which takes place by altering weight values. The following illustration shows a simple ANN –



Types of Artificial Neural Networks: There are two Artificial Neural Network topologies − FeedForward and Feedback.

FeedForward ANN: In this ANN, the information flow is unidirectional. A unit sends information to another unit from which it does not receive any information. There are no feedback loops. They are used in pattern generation/recognition/classification. They have fixed inputs and outputs.

FeedBack ANN: Here, feedback loops are allowed. They are used in content-addressable memories.

Applications of Artificial Neural Networks: Neural Networks are regulating some key sectors including finance, healthcare, and automotive. As these artificial neurons function in a way similar to the human brain. They can be used for image recognition, character recognition and stock market predictions. Some diverse applications of artificial neural networks:

- Facial Recognition
- Stock Market Prediction
- Social Media
- Aerospace
- Defence
- Healthcare
- Signature Verification and Handwriting Analysis
- Weather Forecasting

Keras

Introduction: Keras is an open-source high-level Neural Network library, which is written in Python is capable enough to run on Theano, TensorFlow, or CNTK. It was developed by one of the Google engineers, Francois Chollet. It is made user-friendly, extensible, and modular for facilitating faster experimentation with deep neural networks. It not only supports Convolutional Networks and Recurrent Networks individually but also their combination.

It cannot handle low-level computations, so it makes use of the Backend library to resolve it. The backend library act as a high-level API wrapper for the low-level API, which lets it run on TensorFlow, CNTK, or Theano.

Keras Features

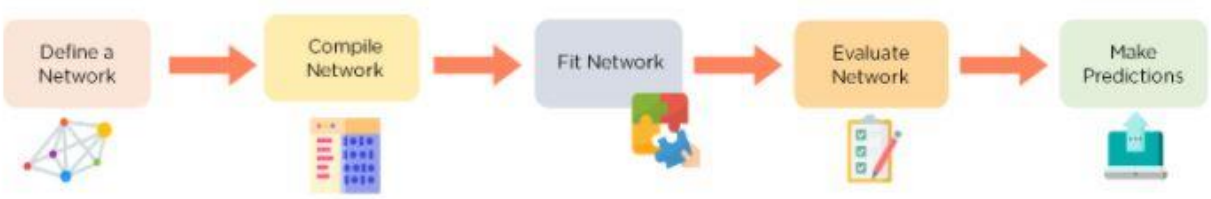
- Keras is an API that was made to be easy to learn for people. Keras was made to be simple. It offers consistent & simple APIs, reduces the actions required to implement common code, and explains user error clearly.
- Prototyping time in Keras is less. This means that your ideas can be implemented and deployed in a shorter time. Keras also provides a variety of deployment options depending on user needs.
- Languages with a high level of abstraction and inbuilt features are slow and building custom features in then can be hard. But Keras runs on top of TensorFlow and is relatively fast. Keras is also deeply integrated with TensorFlow, so you can create customized workflows with ease.
- The research community for Keras is vast and highly developed. The documentation and help available are far more extensive than other deep learning frameworks.
- Keras is used commercially by many companies like Netflix, Uber, Square, Yelp, etc which have deployed products in the public domain which are built using Keras.

Apart from this, Keras has features such as :

- It runs smoothly on both CPU and GPU.

- It supports almost all neural network models.
- It is modular in nature, which makes it expressive, flexible, and apt for innovative research.

How to Build a Model in Keras: The below diagram shows the basic steps involved in building a model in Keras



1. **Define a network:** In this step, you define the different layers in our model and the connections between them. Keras has two main types of models: Sequential and Functional models. You choose which type of model you want and then define the dataflow between them.
2. **Compile a network:** To compile code means to convert it in a form suitable for the machine to understand. In Keras, the model.compile() method performs this function. To compile the model, we define the loss function which calculates the losses in our model, the optimizer which reduces the loss, and the metrics which is used to find the accuracy of our model.
3. **Fit the network:** Using this, we fit our model to our data after compiling. This is used to train the model on our data.
4. **Evaluate the network:** After fitting our model, we need to evaluate the error in our model.
5. Make Predictions: We use model.predict() to make predictions using our model on new data.

Applications of Keras

- Keras is used for creating deep models which can be productized on smartphones.
- Keras is also used for distributed training of deep learning models.
- Keras is used by companies such as Netflix, Yelp, Uber, etc.
- Keras is also extensively used in deep learning competitions to create and deploy working models, which are fast in a short amount of time.

Reference: https://www.javatpoint.com/keras

Tensorflow

Introduction: TensorFlow is an open-source machine learning library developed by Google that is widely used for various machine learning and deep learning tasks. It provides a flexible and efficient framework for building and training machine learning models, particularly neural networks. TensorFlow is often used for assignments in machine learning and deep learning courses because of its popularity, extensive documentation, and robust community support. TensorFlow is based on the concept of tensors, which are multi-dimensional arrays. These tensors can represent data, such as numbers, images, or sequences, and operations performed on them.

Computational Graph: TensorFlow uses a computational graph to define and execute operations. In this graph, nodes represent mathematical operations, and edges represent the flow of data (tensors) between these operations. This graph allows for efficient execution and optimization of operations, especially when training large neural networks.

High-Level and Low-Level APIs: TensorFlow provides both high-level and low-level APIs, making it suitable for both beginners and experienced developers. High-level APIs like Keras offer an easy-to-use interface for building and training neural networks with fewer lines of code. Low-level APIs give you more control over model architecture and training processes, making them suitable for advanced users who need customization.

Application of Tensorflow

Model Building: We can create and define machine learning models using TensorFlow by stacking layers. These layers can be fully connected (dense), convolutional, recurrent, and more, allowing you to design various neural network architectures.

Training Models: TensorFlow provides tools for training models using optimization techniques like gradient descent and its variants. During training, you can define loss functions, monitor metrics, and adjust model parameters to minimize the loss and improve performance.

Data Processing: TensorFlow includes utilities for data preprocessing, such as data augmentation, normalization, and batching, which are essential for preparing datasets for machine learning tasks.