CSE 4554
Machine Learning Lab

Experiment No: 1
Name of the experiment: Introduction to Python for Machine Learning

Hasan Mahmud, Ph.D.
Professor, Department of CSE, IUT
Md. Tanvir Hossain Saikat
Junior Lecturer, Department of CSE, IUT

September 10, 2024

Contents

1 Objectives

- To learn the basics of Python Programming which mainly focuses in ML
- Introduction of the Python Environments, Editors
- Introducing the Python Library Numpy
- Applying Numpy in different tasks of manipulating Array
- Using Numpy to store the IRIS dataset

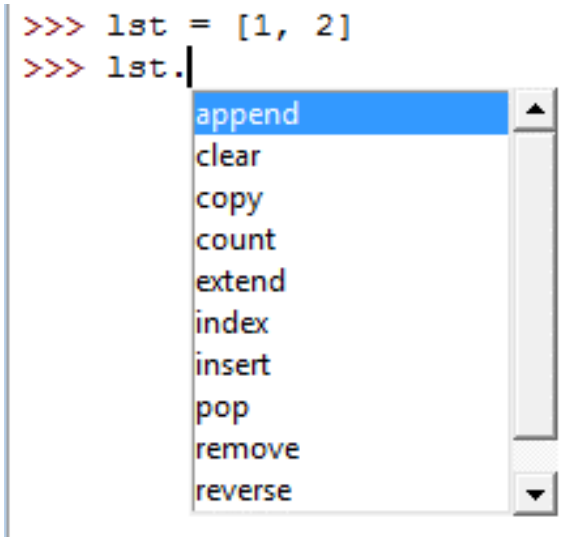
2 Problem Discussion

2.1 Introduction to Python

Get to know about basic python:

I. Python List and tuple, function calling, if else scope, loops and many more
`https://www.programiz.com/python-programming/list`

II. Variable name followed by dot then press Tab to see the available option



III. Negative indexing, which counts backward from the end of the sequence. The expression seq[-1] yields the last element, seq[-2] yields the second to last, and so on.

Listing 1: Negative indexing

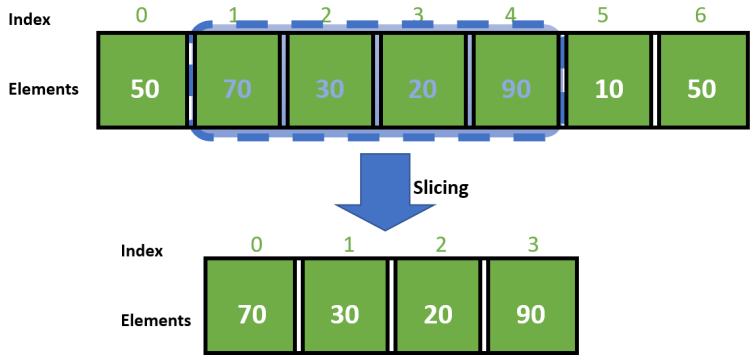
```
classmates=["Alejandro", "Ed", "Kathryn", "Presila", "Sean", "Pet"]
classmates[-5]
# 'Ed'
word = "Alphabet"
word[-3]
# 'b'
```

IV. Slicing. Like with indexing, we use square brackets ([ ]) as the slice operator, but instead of one integer value inside we have two, separated by a colon (:)

Listing 2: Slicing

```
singers="Peter, Paul"
singers[0:5]
# 'Peter'
classmates=["Alejandro", "Ed", "Kathryn", "Presila", "Sean", "Pet"]
classmates[2:4]
# ['Kathryn', 'Presila']
```

The operator [n:m] returns the part of the sequence from the n'th element to the m'th element, including the first but excluding the last.



Lst[1:5]

If you omit the first index (before the colon), the slice starts at the beginning of the string. If you omit the second index, the slice goes to the end of the string. Thus:

Listing 3: Slicing

```
fruit="banana"
fruit[:3]
# 'ban'
fruit[3:]
# 'ana'
```

What do you think s[:] means? It means all the elements
Negative indexes are also allowed, so

Listing 4: Slicing

```
fruit[-2:]
# 'na'
classmates[:-2]
# ['Alejandro','Ed','Kathryn','Presila']
```

V. To handle multidimensional array Numpy array library is used, however library need to import first then using the "as" command you can create an instance for the librar "Import numpy as np"
`https://jakevdp.github.io/PythonDataScienceHandbook/02.02-the-basics-of-numpy-array html`

Listing 5: Numpy indexing

```
    #indexing numpy array
>>> a = np.array([[1, 2, 3], [4, 5, 6]])
>>> a[1, 2]
6
>>> a[1]
array([4, 5, 6])
>>> a[:, 2]
array([3, 6])
>>> a[:, 1:3]
array([[2, 3],
       [5, 6]])
>>> b = a > 2
>>> b
array([[False, False,  True],
       [ True,  True,  True]], dtype=bool)
>>> a[b]
array([3, 4, 5, 6])
```

2.2 Introduction to Numpy

NumPy is one of the fundamental packages for scientific computing in Python. It contains functionality for multidimensional arrays, high-level mathematical functions such as linear algebra operations and the Fourier transform, and pseudorandom number generators. In scikit-learn, the NumPy array is the fundamental data structure. scikit-learn takes in data in the form of NumPy arrays. Any data you're using will have to be converted to a NumPy array. The core functionality of NumPy is the ndarray class, a multidimensional (n-dimensional) array. All elements of the array must be of the same type. NumPy arrays form the core of nearly the entire ecosystem of data science tools in Python. Let us practice some example Numpy codes.

I. Numpy array:
a = [1,2,3,4,5,6,7,8,9]
It is a list
A = np.array([1,2,3,4,5,6,7,8,9])
It is a numpy array

Why do we need numpy array when we have list?

II. Arange:
numpy.arange(start, stop, step, dtype) : Creats a numpy array with only take the elements after each step size
A = np.arange(0,10,2)
A is array([0, 2, 4, 6, 8])

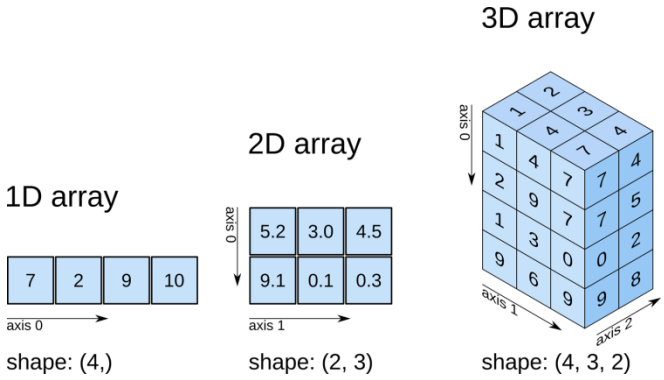
III. Shape:
Shape is an attribute for np array. When a default array, say for example A is called with shape, here is how it looks.
A = [1, 2, 3, 4, 5, 6, 7, 8, 9]
A.shape
–> (9,)

IV. Reshape:
A is a rank 1 matrix(array), where it just has 9 elements in a row. reshape() that changes

the dimensions of your original matrix into your desired dimension.
A.reshape(1,9)

$$\begin{bmatrix}[1]\\[2]\\[3]\\[4]\\[5]\\[6]\\[7]\\[8]\\[9]\end{bmatrix}$$

Two square brackets in the beginning indicate that. [[1, 2, 3, 4, 5, 6, 7, 8, 9]] is a potentially multi-dim matrix as opposed to [1, 2, 3, 4, 5, 6, 7, 8, 9].
To understand array Shape write command variablename.shape or to change dimension write variablename.reshape(x value,y value,z value)



V. Numpy array attributes

Listing 6: Numpy array attributes

```
import numpy as np
np.random.seed(0) # to generate reproducible numbers

x1 = np.random.randint(10, size=6) # one dimensional array of size 6
    from 0 to 9
x2 = np.random.randint(10, size=(3,4)) # two dimensional array
# similarly, you can create a three dimensional array
```

Each array has attributes ndim (the number of dimensions), shape (the size of each dimension), and size (the total size of the array):

Listing 7: Numpy array attributes

```
print("x3 ndim:", x3.ndim)
print("x3 shape:", x3.shape)
print("x3 size:", x3.size)
```

VI. One dimensional sub arrays

Listing 8: One dimensional subarrays

```
x=np.arange(10)
x[::2] # every other element
array([0,2,4,6,8])
x[1::2] # every other element starting at index 1
x[::-1] # all elements reversed
```

VII. Multi-dimensional sub arrays

Listing 9: Multi-dimensional subarrays

```
x = np.array([[12, 5, 2, 4],
              [7, 6, 8, 8],
              [1, 6, 7, 7]])
x2[:2, :3] # two rows, three columns
print(x2[0]) # equivalent to x2[0,:]
```

VIII. Creating arrays: np.zeros((n,m)) returns an n x m matrix that contains zeros. It's as simple as that

Listing 10: Creating arrays with zeros

```
np.zeros((4,3))
# array([[12, 5, 2, 4],
#        [7, 6, 8, 8],
#        [1, 6, 7, 7]])
```

IX. Concatenation of arrays

Listing 11: Concatenation of arrays

```
x=np.array([1,2,3]
y=np.array([3,2,1]
np.concatenate([x,y])
```

X. **Loading a CSV File**

Listing 12: Loading a csv file

```
white_wines = np.genfromtxt("winequality-white.csv", delimiter=",",
    skip_header=1)
```

- `np.genfromtxt`: This function loads data from a text file, offering more control over handling missing data compared to `numpy.loadtxt`. - `"winequality-white.csv"`: The path to the CSV file to be loaded. In this case, it contains data on white wine quality. - `delimiter=","`: Specifies that the file is comma-separated, which is typical for CSV files. - `skip_header=1`: Skips the first row, often used to ignore the header (column names).

XI. **Handling Missing Data and Enforcing Float Type**

Listing 13: Handling missing data

```
data = np.genfromtxt(path, delimiter=',', skip_header=1, filling_values
    =-999, dtype='float')
```

- `path`: A variable representing the path to the file. - `delimiter=','`: Specifies that the file is comma-separated. - `skip_header=1`: Skips the first row of the file. - `filling_values=-999`: Replaces any missing values with `-999`, which can be used as a placeholder for missing data. - `dtype='float'`: Ensures that the data is loaded as floating-point numbers.

XII. **Automatic Data Type Detection**

Listing 14: Automatic data type detection

```
data2 = np.genfromtxt(path, delimiter=',', skip_header=1, dtype=None)
```

- `delimiter=','`: Specifies that the file is comma-separated. - `skip_header=1`: Skips the first row of the file. - `dtype=None`: Automatically determines the appropriate data type for each column. This is useful when dealing with mixed data types, such as text and numerical data.

2.3 Introduction to Pandas

Pandas is an open-source data analysis and manipulation tool built on top of the Python programming language. It provides high-level data structures and functions designed to make data analysis fast and easy. In the context of machine learning, Pandas is crucial for handling structured data, which is often presented in the form of tabular data (i.e., rows and columns).

Machine learning workflows typically begin with data loading, preprocessing, exploration, and manipulation, all of which can be efficiently handled using Pandas. Pandas primarily uses two data structures:

- **Series:** A one-dimensional labeled array capable of holding any data type.
- **DataFrame:** A labeled data structure with columns of potentially different types.

Why Pandas is Important for Machine Learning

- **Data Cleaning:** Handling missing or inconsistent data, which is a common issue in real-world datasets.
- **Data Transformation:** Reformatting and transforming data to make it suitable for machine learning models.
- **Exploratory Data Analysis (EDA):** Analyzing data distributions, relationships, and summarizing statistics before applying machine learning algorithms.
- **Integration with Other Libraries:** Pandas integrates smoothly with other machine learning libraries, such as NumPy, Scikit-learn, TensorFlow, and others.

Sample Code: Pandas in Action

In this section, we demonstrate how to use Pandas for basic data manipulation and preprocessing tasks.

1. Importing and Exploring Data

The first step in working with data is to load it into a Pandas DataFrame. Let's assume we have a CSV file named `data.csv`.

Listing 15: Importing and exploring a dataset

```
import pandas as pd

# Load the dataset
df = pd.read_csv('data.csv')

# Display the first 5 rows of the dataset
print(df.head())

# Get summary statistics
print(df.describe())

# Check for missing values
print(df.isnull().sum())
```

2. Handling Missing Data

Handling missing data is a common task in machine learning workflows. Pandas provides methods to either drop missing data or fill them with appropriate values.

Listing 16: Handling missing data

```
# Drop rows with missing values
df_cleaned = df.dropna()

# Fill missing values with the mean of the respective column
df_filled = df.fillna(df.mean())
```

3. Feature Selection and Data Transformation

Pandas is also used to select features (columns) and transform them for machine learning models. In this example, we select specific columns for the model.

Listing 17: Feature selection and data transformation

```
# Select specific columns (features) for machine learning
selected_features = df[['age', 'salary', 'years_of_experience']]

# Standardize numerical data
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaled_features = scaler.fit_transform(selected_features)

# Convert the scaled features back into a DataFrame
df_scaled = pd.DataFrame(scaled_features, columns=['age', 'salary', '
    years_of_experience'])

print(df_scaled.head())
```

4. Data Aggregation and Grouping

Pandas makes it easy to perform data aggregation and grouping tasks. For example, you can group data by a certain feature and calculate summary statistics.

Listing 18: Data aggregation and grouping

```
# Group by a specific column and calculate the mean of each group
grouped_data = df.groupby('department').mean()

print(grouped_data)
```

3 Tasks

- Numpy
  - **Task 1:**
    Get all the items between 5 and 10 from a. a = np.array([2, 6, 1, 9, 10, 3, 27])
  - **Task 2:**
    Create an ndarray x of shape= (5, 4) with random numbers – Each of the 5 rows represents a sample with 4 features

- **Task 3:**
  Create a flat ndarray y of shape=(5,), whose elements are 0 and 1 – Each element is the class label of the corresponding sample in x [Hint: use np.random.rand(...) and np.random.randint(...)]
- **Task 4:**
  Define a function extract_subset(x, y, y0) that takes as input: – the feature matrix x, and the labels y from the previous exercise – a target class y0 (i.e., either y0=0 or y0=1) and returns a feature matrix containing only samples belonging to y0
- **Task 5:**
  Import a dataset (IRIS dataset) by keeping the text and numbers intact.
- **Task 6:**
  Find the mean, median and standard deviation of the 1st column of the given dataset. Give statistical summary of the dataset as many as possible.

- Pandas : Use "sample_data/california_housing_test.csv" from google colab to do these tasks
  - **Task 1:**
    Loading & Inspecting Data
    * Load a CSV file into a Pandas DataFrame.
    * Display the first 10 rows.
    * Check the data types of each column.
    * Print summary statistics of numerical columns.
  - **Task 2:**
    Handling Missing Data
    * Identify columns with missing values and count them.
    * Remove rows with missing values.
    * Fill missing values with the mean or mode.
  - **Task 3:**
    Suppose you are an investor and want to invest in housing properties. Use this dataset to make decision on the location of investing.

4 Submission

Submit Your Google Colab notebook in the classroom with the following naming format <Student_id>_Lab<Lab_id>.