Python Data Structures

StartOnAI Tutorial 6:

This is a background tutorial that discusses the main Python Data Structures you need to know for Machine Learning!

1. What is the topic?

Data structures are a particular way of organizing data in a computer so it can be efficiently used in the future. Data structures are extremely useful to solve problems where you have to store data and use it later. More specifically, in this tutorial, we will cover python data structures, which are used in machine learning. By learning these data structures, your road to actually applying machine learning will be on its way.

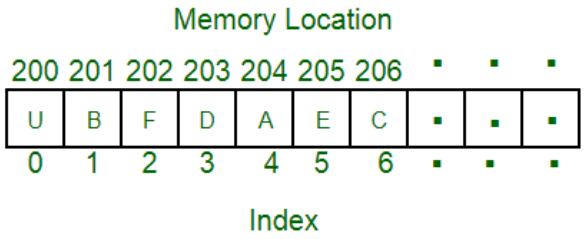
2. Main concepts to know

Before we start, let us start by talking about common libraries used for data structures in python. The most common libraries are Pandas, NumPy, and TensorFlow. These libraries provide a lot of data structures and other resources which allow us to perform Machine Learning very fast and easily. In this lecture, we will include a description of each algorithm, some of its applications along with it being implemented.

Arrays:

Let us start with Arrays also known as Lists in Python. An array is a data structure that stores data in continuous memory locations. In an array, you store items of the same data type together. This makes it easier to calculate the position of each element, generally known as index positions. Looking at the above image, we can clearly notice that each element within the array can be uniquely identified and located with the above index

positions.



Code Walkthrough:

1. In this walkthrough we will talk about the important functions of an array(also known as a list) in Python, but let's first initialize an array and let's call it *colors.* Inside colors, we put in a list of common colors and then we print it out.

```
colors = ["Blue", "Red", "Green", "Orange", "Black", "Yellow"]
print(colors)
```

```
['Blue', 'Red', 'Green', 'Orange', 'Black', 'Yellow']
```

2. Now let us start talking about how to access elements in an array. As stated before each value in an array has an index position starting from 0. That is why the below code prints out *'Red'* because *Blue* has index position 0.

```
colors[1]
```

```
'Red'
```

3. There is also negative indexing in Python for arrays. A negative index allows us to go from the back of the array to the front. The only catch here is that it starts from -1 instead of 0 because there cannot be any negative 0 value.

```
colors[-1]
```

```
'Blue'
```

4. We can also print out a range of values. We specify the range we want through the index positions.

```
colors[3:6]
```

```
['Green', 'Orange', 'Black']
```

5. We can also iterate through an array using a for loop. We do *for color* meaning for

each individual color in the list colors, print out that specific color.

```
for color in colors:
    print(color)
```

```
Blue
Red
Green
Orange
Black
Yellow
```

6. We can also append items to an array using the *.append()* function. In this example what we do is we append Purple to the list of colors, and then when we print it out it shows that colors has been printed out.

```
colors.append("Purple")
colors
```

```
['Blue', 'Red', 'Green', 'Orange', 'Black', 'Yellow', 'Purple']
```

7. Lastly, we are going to cover how to get the length of the array colors. We do this using the *len()* function which outputs an integer with the length

```
print(len(colors))
```

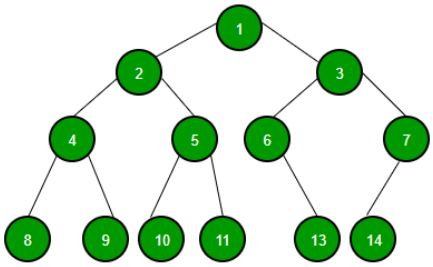
```
8
```

Trees:

A tree is a type of nonlinear data structure. This is different from data structures such as stacks, queues, and arrays which are linear data structures. When we visualize this data structure it looks particularly like a Tree so that's how the name comes about. Each tree has a root value and subtrees of children with a parent node, represented by linked nodes.

Let us use an image to better visualize it.

The number 1 in this scenario is known as the root node. Nodes 4 and 5 are known as sibling nodes, while number 2 is the parent node of nodes 4 and 5. Nodes 4 and 5 individually are known as child nodes. Some other terminology is path, which refers to the sequence of nodes along the edges of trees. The tree structure has many different applications to machine learning but also to other data structures. We can create a binary tree which means that each parent node only has two child nodes. We can also apply algorithms such as the binary search on the tree structure.



NumPy Arrays, Matrices, Vectors:

NumPy is short for Numerical Python and is commonly used for numerical computing in Python. NumPy has become so popular that it has become a common form of data exchange in Python, and many libraries take NumPy arrays as input.

NumPy Arrays: One of NumPy's key features is its N-dimensional array object, also known as an array, which is an extremely fast container for large data sets. It is commonly 10 to 100 times faster than the standard list in python. The ndarray allows you to easily scale our data and allows us to perform the standard operations extremely easily. Let's get started importing NumPy.

```
import numpy as np
```

```
my_arr = np.arange(1000000)
```

```
my_list = list(range(1000000))
```

```
#Using NumPy Array
%time for _ in range(10): my_arr2 = my_arr * 2

#Using List
%time for _ in range(10): my_list2 = [x * 2 for x in my_list]
```

```
Wall time: 25 ms
Wall time: 897 ms
```

As you can see, computing a NumPy array takes ~25 ms, while a normal list takes around

Code Walkthrough:

1. Now, let's get started learning more about ndarrays! First let us import NumPy as the variable np. Next we can initialize a regular list in Python and then we can convert that into a ndarray.

```
import numpy as np
data = [1, 2, 3, 4, 5, 6]

arr1 = np.array(data)

arr1
```

```
array([1, 2, 3, 4, 5, 6])
```

2. We can create multidimensional arrays with lists in Python and we can also convert those into ndarrays

```
data2 = [[1,2,3,4], [5,6,7,8]]

arr2 = np.array(data2)

arr2
```

```
array([[1, 2, 3, 4],
       [5, 6, 7, 8]])
```

3. In a NumPy array we can also print out its dimensions and its shape. In this scenario, arr2 is a two-dimensional array because we have nested arrays. And since we have nested arrays our shape will be *(2,4)* because of the four elements within each array.

```
print(arr2.ndim)
print(arr2.shape)
```

```
2
(2, 4)
```

4. We can also check the data type that is being used within the ndarray. In this case, the data type in our area was *int32*.

```
arr2.dtype
```

```
dtype('int32')
```

5. We can also fill up ndarrays with zeros, creating basically empty arrays. You can also make these multidimensional arrays.

```
np.zeros(5)
```

```
array([0., 0., 0., 0., 0.])
```

```
np.zeros((3, 6))
```

```
array([[0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.]])
```

6. NumPy Arrays can also normally be filled with garbage values and this happens when you use the command *empty()*. It often returns garbage values which can often easily be replaced.

```
np.empty((2, 3, 2))
```

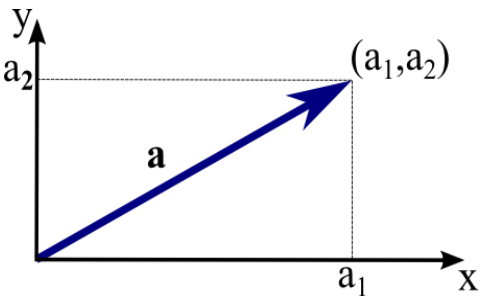
```
array([[[1.34121753e-311, 3.16202013e-322],
        [0.00000000e+000, 0.00000000e+000],
        [1.89146896e-307, 1.04035900e-042]],

       [[1.33693780e+165, 2.63345801e-052],
        [4.12365271e-061, 4.47165652e-062],
        [1.84744733e+184, 9.06534867e-043]]])
```

Matrices: A Matrix is a two-dimensional data structure(similar to a DataFrame) and with all of its elements being of the same data type(Just like an array). It is very important to learn about matrices as many Python functions return matrices. An example of this is in Computer Vision, while you are analyzing an image and look at the colors in that image, it will return a matrix of values between 0-255 which represents the colors. We can create Matrices using NumPy's multidimensional ndarrays, however, you can also use other

modules such as SciPy, and Patsy to create matrices.

Vectors: Vectors are extremely important in linear algebra, and machine learning. Vectors are used throughout machine learning and are often used to process the dependent variable. A vector is a tuple of one or more values called scalars. They can also be thought of as a list of values which are the same data type. In a geometric sense, vectors can be thought of as a line from the origin in vector space which has both a direction and a magnitude. An example of usage of a vector is when they are used to store features. Often there is a feature vector in machine learning that corresponds to some features. If we take breast cancer, for example, some features that would be in a feature vector could be age, height, weight, hair color, etc.



Pandas DataFrame, Series:

DataFrames and Series are data structures in the Pandas Library which are used for storing data.

Series: It is a one-dimensional data structure. It is quite similar to an array and it contains a sequence of values, along with an associated array of data labels called an index. If an index is not specified for the data, a default index is created. The index consists of integers 0 through N-1(N == Length of the Data). Through this, you can get the array representation and index object of the Series.

DataFrame: A DataFrame is a rectangular table of data and contains an ordered collection of columns, which can all be a different data type(int, long, float, string, boolean, etc.). The DataFrame is similar to a row and column index and can be thought of like a dictionary. The data is stored in a two-dimensional block. There are many ways to construct a DataFrame from reading from a CSV file(common in Machine Learning) or by using NumPy Arrays.

Code Walkthrough:

1. Let's get started with our Pandas DataFrame walkthrough! Before we start, make sure to import *Pandas* and *NumPy*. After that I just created a dictionary, which mapped the value *Country* to some large countries in the world and the value *Population* to the population of those countries in order. Then I converted this dictionary into a DataFrame using *pd.DataFrame(data)*

```
import pandas as pd
import numpy as np

data = {'Country' : ['America', 'China', 'India', 'Brazil'],
        'Population' :[331002651, 1439323776, 1380004385, 212559417]}

frame = pd.DataFrame(data)
```

2. Now, using the *frame.head()* command I am able to print the first five items in my DataFrame. Although this DataFrame only contains 2 columns and 4 values in each of them, in Machine Learning there are often 30-50 columns, and 100's of features in each.

```
frame.head()
```

	Country	Population
0	America	331002651
1	China	1439323776
2	India	1380004385
3	Brazil	212559417

3. In a DataFrame we can often switch around the columns and that is what I did here. I simply switched the population and the country columns.

```
pd.DataFrame(data, columns=['Population', 'Country'])
```

	Population	Country
0	331002651	America
1	1439323776	China
2	1380004385	India
3	212559417	Brazil

4. Lastly, we can output the result of just one column of a DataFrame using the *frame.Population* command. In this case, we can replace *.Population with any other feature* that is a part of our DataFrame, and it will output the values that are part of that column.

```
frame.Population
```

```
0     331002651
1    1439323776
2    1380004385
3     212559417
Name: Population, dtype: int64
```

TensorFlow Graphs(Data Flow), Edges(Tensors, Special Edges):

TensorFlow is an extremely important Python Library and is used for many machine learning projects. Here I will give just a brief introduction to TensorFlow's data structures, for a more in-depth tutorial please check out the *TensorFlow I* and *TensorFlow II*.

Data Flow Graphs: In TensorFlow, computation is performed using Data Flow Graphs. Every node in the data flow graph represents a mathematical operation and each edge is a tensor(multi-dimensional data set), which we use to perform operations. In TensorFlow, there are also many different data types which are used for mathematical computation such as constants(floats, ints,etc.), variables(constants) and sessions.

Edges: Edges in data flow graphs are represented by the data produced by the computation. Edges can be grouped into two primary categories. Regular edges transfer tensors where the output for one operation becomes the input for another. Next, there are special edges, which are used to where there is an ordering set between the nodes, so the first node is completed then the second node starts. In this tutorial, we will discuss tensors.

Tensors: Tensors are one of the basic data structures in TensorFlow, and represent the connected edges of a data flow graph. A tensor is simply just a multidimensional array. There are 3 parameters to the tensor, rank, shape, and type. The rank describes the number of dimensions of the tensor, the shape is the number of rows and columns, and the type is simply just the data type of the tensor. To build a tensor we have to use a ndarray so we can quickly implement one from NumPy.

3. Concepts useful to Machine Learning

Learning these data structures will allow you to be extremely efficient when you start conducting your own machine learning research. For example, without Data Structures such as DataFrames, and Series conducting your machine learning research

would be extremely difficult. Along with this understanding how Data Structures such as Trees, and Vectors work and how they are implemented will be extremely important when you start learning machine learning algorithms such as Decision Trees and Support Vector Machines because the base concepts will allow you to understand those concepts a lot better. For example, a Decision Tree is simply just a tree with each branch representing the outcome of the test, and each leaf node represents a feature. Machine Learning datasets on websites such as the UCI Machine Learning Repository or image datasets such as Open Images are normally extremely long and rich with information. And to store all this data we use Data Structures such as DataFrames, and matrices. Then after cleaning up and understanding the data we start using the aforementioned machine learning algorithms and we test the accuracy of our model. Learning the in's and outs of these data structures will be a great benefit to you when you start making your own machine learning projects!

4. Additional Resources

- Geeks for Geeks Data Structures
    - Check Geeks for Geeks for reviewing all the different types of data structures
- Python for Data Analysis: Wrangling with Pandas, NumPy, and IPython'
    - Check out this book for an in depth review of Pandas and NumPy. This book has a lot of great information and by reading this book your Pandas and NumPy abilities will go to the next level.
- NumPy Tutorial
    - Check this NumPy tutorial to learn about NumPy, python data structures and Matplotlib
- TensorFlow Data Structures
    - To learn about the TensorFlow data structures in depth check out the TensorFlow website.
- Complete Pandas Data Science Tutorial
    - By watching Keith Galli's one hour Pandas tutorial, you will definitely understand how to properly apply Pandas to Data Science. It has segments about how to read data from a csv, and sorting and filtering through columns. Be sure to check it out.

5. References

Geeks for Geeks. (n.d.). Array Data Structure. Retrieved April 9, 2020, from https://www.geeksforgeeks.org/array-data-structure/

Geeks for Geeks. (n.d.). Binary Tree Data Structure. Retrieved April 9, 2020, from https://www.geeksforgeeks.org/binary-tree-data-structure/

Nykamp, D. Q. (n.d.). The coordinates of a vector in two dimensions. Retrieved April 9, 2020, from https://mathinsight.org/image/vector_2d_coordinates