Author - NITIN

Data Science and Machine Learning

Task – 1

Numpy-

NumPy is a Python package providing fast, flexible, and expressive data structures designed to make working with 'relationa' or 'labeled' data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python.

NumPy Basics

Operator	Description
np.array([1,2,3])	1d array
np.array([(1,2,3),(4,5,6)])	2d array
np.arange(start,stop,step)	range array

Arrays

A numpy array is a grid of values, all of the same type, and is indexed by a tuple of nonnegative integers. The number of dimensions is the *rank* of the array; the *shape* of an array is a tuple of integers giving the size of the array along each dimension.

We can initialize numpy arrays from nested Python lists, and access elements using square brackets:

Some common functions –

Common Functions:

np.zeros()	np.append()	np.insert()
np.random()	np.resize()	np.split()
np.reshape()	np.squeeze()	np.concatenate()
np.flatten()	np.broadcast()	variable_name.T

Math Functions:

np.sin()	np.cos()	np.tan()	
np.arcsin()	np.arccos()	np.arctan()	
np.degrees()	np.around()	np.ceil()	np.floor()

Linear Algebra:

np.dot()	np.vdot()	np.determinant()
np.inv()	np.inner()	np.matmul()
np.add()	np.inner()	np.multiply()
np.add()	np.divide()	np.subtract()
np.mod()	np.power()	np.reciprocal()

CODE –

```
import numpy as np

a = np.array([1, 2, 3])   # Create a rank 1 array
print(type(a))            # Prints "<class 'numpy.ndarray'>"
print(a.shape)            # Prints "(3,)"
print(a[0], a[1], a[2])   # Prints "1 2 3"
a[0] = 5                  # Change an element of the array
print(a)                  # Prints "[5, 2, 3]"

b = np.array([[1,2,3],[4,5,6]])    # Create a rank 2 array
print(b.shape)                     # Prints "(2, 3)"
print(b[0, 0], b[0, 1], b[1, 0])   # Prints "1 2 4"
```

Numpy also provides many functions to create arrays:

```
import numpy as np

a = np.zeros((2,2))   # Create an array of all zeros
print(a)              # Prints "[[ 0.  0.]
                      #          [ 0.  0.]]"

b = np.ones((1,2))    # Create an array of all ones
print(b)              # Prints "[[ 1.  1.]]"

c = np.full((2,2), 7)  # Create a constant array
```

Array indexing

Numpy offers several ways to index into arrays.

Slicing: Similar to Python lists, numpy arrays can be sliced. Since arrays may be multidimensional, you must specify a slice for each dimension of the array:

```
import numpy as np

# Create the following rank 2 array with shape (3, 4)
# [[ 1  2  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]]
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

# Use slicing to pull out the subarray consisting of the first 2 rows
# and columns 1 and 2; b is the following array of shape (2, 2):
# [[2 3]
#  [6 7]]
b = a[:2, 1:3]

# A slice of an array is a view into the same data, so modifying it
# will modify the original array.
print(a[0, 1])   # Prints "2"
b[0, 0] = 77     # b[0, 0] is the same piece of data as a[0, 1]
print(a[0, 1])   # Prints "77"
```

Integer array indexing: When you index into numpy arrays using slicing, the resulting array view will always be a subarray of the

original array. In contrast, integer array indexing allows you to construct arbitrary arrays using the data from another array. Here is an example:

```
import numpy as np

a = np.array([[1,2], [3, 4], [5, 6]])

# An example of integer array indexing.
# The returned array will have shape (3,) and
print(a[[0, 1, 2], [0, 1, 0]])  # Prints "[1 4 5]"

# The above example of integer array indexing is equivalent to this:
print(np.array([a[0, 0], a[1, 1], a[2, 0]]))  # Prints "[1 4 5]"

# When using integer array indexing, you can reuse the same
# element from the source array:
print(a[[0, 0], [1, 1]])  # Prints "[2 2]"

# Equivalent to the previous integer array indexing example
print(np.array([a[0, 1], a[0, 1]]))  # Prints "[2 2]"
```

One useful trick with integer array indexing is selecting or mutating one element from each row of a matrix:

```
import numpy as np

# Create a new array from which we will select elements
a = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])

print(a)  # prints "array([[ 1,  2,  3],
          #               [ 4,  5,  6],
          #               [ 7,  8,  9],
          #               [10, 11, 12]])"

# Create an array of indices
b = np.array([0, 2, 0, 1])

# Select one element from each row of a using the indices in b
print(a[np.arange(4), b])  # Prints "[ 1  6  7 11]"

# Mutate one element from each row of a using the indices in b
a[np.arange(4), b] += 10

print(a)  # prints "array([[11,  2,  3],
          #               [ 4,  5, 16],
          #               [17,  8,  9],
          #               [10, 21, 12]])
```

Boolean array indexing: Boolean array indexing lets you pick out arbitrary elements of an array. Frequently this type of indexing is used to select the elements of an array that satisfy some condition. Here is an example:

```
import numpy as np

a = np.array([[1,2], [3, 4], [5, 6]])
```

```
bool_idx = (a > 2)  # Find the elements of a that are bigger than 2;
                    # this returns a numpy array of Booleans of the same
                    # shape as a, where each slot of bool_idx tells
                    # whether that element of a is > 2.

print(bool_idx)     # Prints "[[False False]
                    #          [ True  True]
                    #          [ True  True]]"

# We use boolean array indexing to construct a rank 1 array
# consisting of the elements of a corresponding to the True values
# of bool_idx
print(a[bool_idx])  # Prints "[3 4 5 6]"

# We can do all of the above in a single concise statement:
print(a[a > 2])     # Prints "[3 4 5 6]"
```

Broadcasting

Broadcasting is a powerful mechanism that allows numpy to work with arrays of different shapes when performing arithmetic operations. Frequently we have a smaller array and a larger array, and we want to use the smaller array multiple times to perform some operation on the larger array.

For example, suppose that we want to add a constant vector to each row of a matrix. We could do it like this:

```
import numpy as np

# We will add the vector v to each row of the matrix x,
# storing the result in the matrix y
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
```

```
v = np.array([1, 0, 1])
y = np.empty_like(x)   # Create an empty matrix with the same shape as x

# Add the vector v to each row of the matrix x with an explicit loop
for i in range(4):
    y[i, :] = x[i, :] + v

# Now y is the following
# [[ 2  2  4]
#  [ 5  5  7]
#  [ 8  8 10]
#  [11 11 13]]
print(y)
```

```
print(c)            # Prints "[[ 7.  7.]
                    #         [ 7.  7.]]"

d = np.eye(2)       # Create a 2x2 identity matrix
print(d)            # Prints "[[ 1.  0.]
                    #         [ 0.  1.]]"

e = np.random.random((2,2))  # Create an array filled with random values
print(e)                     # Might print "[[ 0.91940167  0.08143941]
                             #              [ 0.68744134  0.87236687]]"
```

Pandas –

Pandas is a Python library used for working with data sets. It has functions for analyzing, cleaning, exploring, and manipulating data. The name Pandas' is a reference to both 'Panel Data' and 'Python Data Analysis'.

Some common functions –

• read_csv : read the data from the csv file.

• index_col = None : there is no index i.e. first column is data

• head() : show only first five elements of the DataFrame

• tail() : show only last five elements of the DataFrame

Row and column selection -

Any row or column of the DataFrame can be selected by passing the name of the column or rows. After selecting one from DataFrame, it becomes one-dimensional therefore it is considered as Series.

Filter Data -

Data can be filtered by providing some boolean expression in DataFrame. For example, in below code, movies which released after

1985 are filtered out from the DataFrame ‘titles’ and stored in a new DataFrame

Sorting Sorting-

can be performed using ‘sort_index’ or ‘sort_values’ keywords,

Null values –

Note that, various columns may contains no values, which are usually filled as NaN. For example, rows 3-4 of casts are NaN

String operations-

Various string operations can be performed using ‘.str.

Creating DataFrame: pd.DataFrame({'Yes': [50, 21], 'No': [131, 2]})

Common Functions:

df.describe()	df.mean()	df.unique()	df.head()
df.groupby()	df.sort_values()		df.reset_index()

Functions for Missing Data:

df.nan()	df.dropna()	df.fillna()

Functions for Transformation:

df.groupby()	pd.merge()	pd.concat()	df.stack()
df.duplicated()	df.unstack()	df.drop_duplicated()	
df.rename()	pd.get_dummies()		

Getting Data:

pd.read_csv(‘----’)

pd.DataFrame(json.dumps (json.load(‘---’)) [‘data’], columns=[‘field’])

CODE –

Create a simple Pandas Series from a list:

```
import pandas as pd

a = [1, 7, 2]

myvar = pd.Series(a)

print(myvar)
```

Create a simple Pandas DataFrame:

```
import pandas as pd

data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}

#load data into a DataFrame object:
df = pd.DataFrame(data)

print(df)
```

Load the CSV into a DataFrame:

```
import pandas as pd
```

```
df = pd.read_csv('data.csv')

print(df.to_string())
```

Return a new Data Frame with no empty cells:

```
import pandas as pd

df = pd.read_csv('data.csv')

new_df = df.dropna()

print(new_df.to_string())
```

Get a quick overview by printing the first 10 rows of the DataFrame:

```
import pandas as pd

df = pd.read_csv('data.csv')

print(df.head(10))
```

Returns True for every row that is a duplicate, othwerwise False:

```
print(df.duplicated())
```

Show the relationship between the columns:

```
df.corr()
```

Import pyplot from Matplotlib and visualize our DataFrame:

```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('data.csv')

df.plot()

plt.show()
```