**Day One-Session One**

**Scientific Python Packages Overview**

Beyond Python there are a number of open source libraries generally used to facilitate practical machine learning. In general, these are the main so-called scientific Python libraries we put to use when performing elementary machine learning tasks.

**numpy -** mainly useful for its N-dimensional array objects

**pandas** - Python data analysis library, including structures such as dataframes

**matplotlib** - 2D plotting library producing publication quality figures

**scikit-learn - scikit-learn comes with a few standard datasets, for instance the iris and digits datasets for classification and the boston house prices dataset for regression.** the machine learning algorithms used for data analysis and data mining tasks A good approach to learning these is to cover this material:

**Starting with Numpy**

**(Creating and manipulating array data)**

#load the library and check its version, just to make sure we aren't using an older version

import numpy as np

np.\_\_version\_\_

'1.12.1'

#create a list comprising numbers from 0 to 9

L = list(range(10))

#converting integers to string - this style of handling lists is known as list comprehension.

#List comprehension offers a versatile way to handle list manipulations tasks easily. We'll learn about them in future tutorials. Here's an example.

[str(c) for c in L]

['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']

[type(item) for item in L]

[int, int, int, int, int, int, int, int, int, int]

**Creating Arrays**

Numpy arrays are homogeneous in nature, i.e., they comprise one data type (integer, float, double, etc.) unlike lists.

#creating arrays

np.zeros(10, dtype='int')

array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])

#creating a 3 row x 5 column matrix

np.ones((3,5), dtype=float)

array([[ 1., 1., 1., 1., 1.],

[ 1., 1., 1., 1., 1.],

[ 1., 1., 1., 1., 1.]])

#creating a matrix with a predefined value

np.full((3,5),1.23)

array([[ 1.23, 1.23, 1.23, 1.23, 1.23],

[ 1.23, 1.23, 1.23, 1.23, 1.23],

[ 1.23, 1.23, 1.23, 1.23, 1.23]])

#create an array with a set sequence

np.arange(0, 20, 2)

array([0, 2, 4, 6, 8,10,12,14,16,18])

#create an array of even space between the given range of values

np.linspace(0, 1, 5)

array([ 0., 0.25, 0.5 , 0.75, 1.])

#create a 3x3 array with mean 0 and standard deviation 1 in a given dimension

np.random.normal(0, 1, (3,3))

array([[ 0.72432142, -0.90024075, 0.27363808],

[ 0.88426129, 1.45096856, -1.03547109],

[-0.42930994, -1.02284441, -1.59753603]])

#create an identity matrix

np.eye(3)

array([[ 1., 0., 0.],

[ 0., 1., 0.],

[ 0., 0., 1.]])

#set a random seed

np.random.seed(0)

x1 = np.random.randint(10, size=6) #one dimension

x2 = np.random.randint(10, size=(3,4)) #two dimension

x3 = np.random.randint(10, size=(3,4,5)) #three dimension

print("x3 ndim:", x3.ndim)

print("x3 shape:", x3.shape)

print("x3 size: ", x3.size)

('x3 ndim:', 3)

('x3 shape:', (3, 4, 5))

('x3 size: ', 60)

**Array Indexing**

The important thing to remember is that indexing in python starts at zero.

x1 = np.array([4, 3, 4, 4, 8, 4])

x1

array([4, 3, 4, 4, 8, 4])

#assess value to index zero

x1[0]

4

#assess fifth value

x1[4]

8

#get the last value

x1[-1]

4

#get the second last value

x1[-2]

8

#in a multidimensional array, we need to specify row and column index

x2

array([[3, 7, 5, 5],

[0, 1, 5, 9],

[3, 0, 5, 0]])

#1st row and 2nd column value

x2[2,3]

0

#3rd row and last value from the 3rd column

x2[2,-1]

0

#replace value at 0,0 index

x2[0,0] = 12

x2

array([[12, 7, 5, 5],

[ 0, 1, 5, 9],

[ 3, 0, 5, 0]])

**Array Slicing**

Now, we'll learn to access multiple or a range of elements from an array.

x = np.arange(10)

x

array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

#from start to 4th position

x[:5]

array([0, 1, 2, 3, 4])

#from 4th position to end

x[4:]

array([4, 5, 6, 7, 8, 9])

#from 4th to 6th position

x[4:7]

array([4, 5, 6])

#return elements at even place

x[ : : 2]

array([0, 2, 4, 6, 8])

#return elements from first position step by two

x[1::2]

array([1, 3, 5, 7, 9])

#reverse the array

x[::-1]

array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])

**Array Concatenation**

Many a time, we are required to combine different arrays. So, instead of typing each of their elements manually, you can use array concatenation to handle such tasks easily.

#You can concatenate two or more arrays at once.

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

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

z = [21,21,21]

np.concatenate([x, y,z])

array([ 1, 2, 3, 3, 2, 1, 21, 21, 21])

#You can also use this function to create 2-dimensional arrays.

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

np.concatenate([grid,grid])

array([[1, 2, 3],

[4, 5, 6],

[1, 2, 3],

[4, 5, 6]])

#Using its axis parameter, you can define row-wise or column-wise matrix

np.concatenate([grid,grid],axis=1)

array([[1, 2, 3, 1, 2, 3],

[4, 5, 6, 4, 5, 6]])

Until now, we used the concatenation function of arrays of equal dimension. But, what if you are required to combine a 2D array with 1D array? In such situations, np.concatenate might not be the best option to use. Instead, you can use np.vstack or np.hstack to do the task. Let's see how!

x = np.array([3,4,5])

grid = np.array([[1,2,3],[17,18,19]])

np.vstack([x,grid])

array([[ 3, 4, 5],

[ 1, 2, 3],

[17, 18, 19]])

#Similarly, you can add an array using np.hstack

z = np.array([[9],[9]])

np.hstack([grid,z])

array([[ 1, 2, 3, 9],

[17, 18, 19, 9]])

Also, we can split the arrays based on pre-defined positions. Let's see how!

x = np.arange(10)

x

array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

x1,x2,x3 = np.split(x,[3,6])

print x1,x2,x3

[0 1 2] [3 4 5] [6 7 8 9]

grid = np.arange(16).reshape((4,4))

grid

upper,lower = np.vsplit(grid,[2])

print (upper, lower)

(array([[0, 1, 2, 3],

[4, 5, 6, 7]]), array([[ 8, 9, 10, 11],

[12, 13, 14, 15]]))

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Practicing Scikit learn

scikit-learn comes with a few small standard datasets that do not require to download any file from some external website.

|  |  |
| --- | --- |
| [**load\_boston**](http://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_boston.html#sklearn.datasets.load_boston)([return\_X\_y]) | Load and return the boston house-prices dataset (regression). |
| [**load\_iris**](http://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_iris.html#sklearn.datasets.load_iris)([return\_X\_y]) | Load and return the iris dataset (classification). |
| [**load\_diabetes**](http://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_diabetes.html#sklearn.datasets.load_diabetes)([return\_X\_y]) | Load and return the diabetes dataset (regression). |
| [**load\_digits**](http://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_digits.html#sklearn.datasets.load_digits)([n\_class, return\_X\_y]) | Load and return the digits dataset (classification). |
| [**load\_linnerud**](http://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_linnerud.html#sklearn.datasets.load_linnerud)([return\_X\_y]) | Load and return the linnerud dataset (multivariate regression). |
| [**load\_wine**](http://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_wine.html#sklearn.datasets.load_wine)([return\_X\_y]) | Load and return the wine dataset (classification). |
| [**load\_breast\_cancer**](http://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_breast_cancer.html#sklearn.datasets.load_breast_cancer)([return\_X\_y]) | Load and return the breast cancer wisconsin dataset (classification). |

These datasets are useful to quickly illustrate the behavior of the various algorithms implemented in the scikit. They are however often too small to be representative of real world machine learning tasks.

$ python

>>> from sklearn import datasets

>>> iris = datasets.load\_iris()

>>> digits = datasets.load\_digits()

Data  is a n\_samples, n\_features array.

Practice-1

Digit classification by using SVM

For instance, in the case of the digits dataset, digits.data gives access to the features that can be used to classify the digits samples:

>>>

**>>>** print(digits.data)

[[ 0. 0. 5. ..., 0. 0. 0.]

[ 0. 0. 0. ..., 10. 0. 0.]

[ 0. 0. 0. ..., 16. 9. 0.]

...,

[ 0. 0. 1. ..., 6. 0. 0.]

[ 0. 0. 2. ..., 12. 0. 0.]

[ 0. 0. 10. ..., 12. 1. 0.]]

Print(digits.data.shape)

(1797, 64)

and digits.target gives the ground truth for the digit dataset, that is the number corresponding to each digit image that we are trying to learn:

>>>

**>>>** digits.target

array([0, 1, 2, ..., 8, 9, 8])

digits.images[0]

array([[ 0., 0., 5., 13., 9., 1., 0., 0.],

[ 0., 0., 13., 15., 10., 15., 5., 0.],

[ 0., 3., 15., 2., 0., 11., 8., 0.],

[ 0., 4., 12., 0., 0., 8., 8., 0.],

[ 0., 5., 8., 0., 0., 9., 8., 0.],

[ 0., 4., 11., 0., 1., 12., 7., 0.],

[ 0., 2., 14., 5., 10., 12., 0., 0.],

[ 0., 0., 6., 13., 10., 0., 0., 0.]])

Practice-2