E9 205 Machine Learning for Signal Processing

01-09-2017

Tutorial on Python

Outline

- Basics and control statements
- Functions and lists
- Tuples, Dictionaries, Strings, File i/o
- Modules
- Packages: Numpy, (Scipy, Matplotlib)

Reference:
[1] Stanford university, Course: “Introduction to Scientific Python”
Link: https://web.stanford.edu/~schmit/cme193/resources.html
[2] : PythonLearn : http://www.pythonlearn.com/

How to install Python?

- Anaconda
  - https://store.continuum.io/cshop/anaconda/
- Very easy to install and also comes with a lot of packages.

How to use Python

- There are two ways to use Python:
  - command-line mode: talk directly to the interpreter
  - scripting-mode: write code in a file (called script) and run code by typing in the terminal

python Tutorial1.py

Tutorial1.py----> print "MLSP 2017"

Comments in Python

- Anything after a # is ignored by Python
- Why comment?
  - Describe what is going to happen in a sequence of code
  - Turn off a line of code - perhaps temporarily
  - Easy to evaluate your assignments..

Control statements

- Control statements allow you to do more complicated tasks.
  - if
  - for
  - while

Indentation

- In Python, blocks of code are defined using *indentation*.
- Maintain indent to indicate the *scope* of the block
  - This means that everything indented after an if statement is only executed if the statement is True.
- Reduce indent back to the level of the if statement to indicate the end of the block
  - If the statement is False, the program skips all indented code and resumes at the first line of unindented code

```
if statement:
    # if statement is True, then all code here
    # gets executed but not if statement is False
    print "The statement is true"
    print "Else, this would not be printed"
# the next lines get executed either way
print "Hello, world,"
print "Bye, world!"
```

Control statements

```
if traffic_light == 'green':
    drive()
elif traffic_light == 'orange':
    accelerate()
else:
    stop()
```

```
for i in range(5):
    print i**2,
# 0 1 4 9 16
```

```
i = 1
while i < 100:
    print i**2,
    i += i**2  # a += b is short for a = a + b
# 1 4 36 1764
```

Functions and lists

Functions

- Much like a mathematical function, they take some input and then do something to find the result.
- Start a function definition with the keyword **def**
- Then comes the function name, with arguments in braces, and then a colon--> indentend(body of function) -->return (Specifies o/p)

```
x = 1

def add_one(x):
    x = x + 1  # local x
    return x

y = add_one(x)
# x = 1, y = 2
```

Lists

- Group variables together
- can mix element types
- Access items using square brackets: [ ]

```
myList = [5, 2.3, 'hello']

myList[0]      # 5
myList[2]      # 'hello'
myList[3]      # ! IndexError
myList[-1]     # 'hello'
myList[-3]     # ?
```

Slicing and adding

- Lists can be sliced: [2:5]
- Lists can be multiplied
- Lists can be added

```
myList = [5, 2.3, 'hello']

myList[0:2]       # [5, 2.3]

mySecondList = ['a', '3']

concatList = myList + mySecondList
# [5, 2.3, 'hello', 'a', '3']
```

Lists are mutable

- individual elements can be changed

```
myList = ['a', 43, 1.234]

myList[0] = -3
# [-3, 43, 1.234]

x = 2
myList[1:3] = [x, 2.3]  # or: myList[1:] = [x, 2.3]
# [-3, 2, 2.3]

x = 4
# What is myList now?
```

More control over lists

- `len(xs)`
- `xs.append(x)`
- `xs.count(x)`
- `xs.insert(i, x)`
- `xs.sort()` and `sorted(xs)`: what's the difference?
- `xs.remove(x)`
- `xs.pop()` or `xs.pop(i)`
- `x in xs`

All these can be found in the Python documentation, google: 'python list'

Or using `dir(xs)` / `dir([])`

- Tuples, dictionaries and strings

Tuples

- similar to lists
- Tuples are Immutable-->Unlike lists, we cannot change elements.

```
>>> myTuple = (1, 2, 3)
>>> myTuple[1]
2
>>> myTuple[1:3]
(2, 3)
```

```
>>> myTuple = ([1, 2], [2, 3])
>>> myTuple[0] = [3,4]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not
support item assignment
>>> myTuple[0][1] = 3
>>> myTuple
([1, 3], [2, 3])
```

Dictionaries

- A dictionary is a collection of ***key-value*** pairs.
- An example: the keys are all words in the English language, and their corresponding values are the meanings.

```
>>> d = {}
>>> d[1] = "one"
>>> d[2] = "two"
>>> d
{1: 'one', 2: 'two'}
>>> e = {1: 'one', 'hello': True}
>>> e
{1: 'one', 'hello': True}
```

- Print all key-value pairs of a dictionary

```
>>> d = {1: 'one', 2: 'two', 3: 'three'}
>>> for key, value in d.items():
...     print key, value
...
1 one
2 two
3 three
```

Strings

- Strings hold a sequence of characters.
- We can slice strings just like lists and tuples
- We can turn anything in Python into a string using **str**
- To split a string, for example, into seperate words, we can use split()

```
text = 'Hello, world!\n How are you?'
text.split()
# ['Hello,', 'world!', 'How', 'are', 'you?']
```

```
numbers = '1, 3, 2, 5'
numbers.split()
# ['1,', '3,', '2,', '5']

numbers.split(', ')
# ['1', '3', '2', '5']

[int(i) for i in numbers.split(', ')]
# [1, 3, 2, 5]
```

File I/O

- Interaction with the file system is pretty straightforward in Python.
- Done using file objects
- We can instantiate a file object using open or file
- **f = open(filename, option)**
  - filename: path and filename
  - Option: **'r'** read file, **'w'** write to file, **'a'** append to file
- We need to close a file after we are done: **f.close()**
- read() Read entire line (or first n characters, if supplied)

```
with open('data/text_file.txt', 'r') as f:
    print f.read()
```

Writing to file

- Use write() to write to a file

```
with open(filename, 'w') as f:
    f.write("Hello, {}!\n".format(name))
```

Modules: Importing a module

- We can import a module by using **import**
  - E.g. **import math**
  - We can then access everything in math, for example the square root function, by: **math.sqrt(2)**
- We can rename imported modules
  - E.g. **import math as m**
  - Now we can write **m.sqrt(2)**
- In case we only need some part of a module
  - We can import only what we need using the from ... import ...syntax.
  - E.g. **from math import sqrt**
  - Now we can use **sqrt(2)** directly

- Numpy, Scipy, Matplotlib

Numpy

- Fundamental package for scientific computing with Python
- N-dimensional array object
- Linear algebra, Fourier transform, random number capabilities

```
import numpy as np

A = np.array([[1, 2, 3], [4, 5, 6]])
print A
# [[1 2 3]
#  [4 5 6]]

Af = np.array([1, 2, 3], float)
```

```
np.arange(0, 1, 0.2)
# array([ 0. ,  0.2,  0.4,  0.6,  0.8])

np.linspace(0, 2*np.pi, 4)
# array([ 0.0,  2.09,  4.18, 6.28])

A = np.zeros((2,3))
# array([[ 0.,  0.,  0.],
#        [ 0.,  0.,  0.]])
# np.ones, np.diag
A.shape
# (2, 3)
```

```
np.random.random((2,3))
# array([[ 0.78084261,  0.64328818,  0.55380341],
#        [ 0.24611092,  0.37011213,  0.83313416]])

a = np.random.normal(loc=1.0, scale=2.0, size=(2,2))
# array([[ 2.87799514,  0.6284259 ],
#        [ 3.10683164,  2.05324587]])

np.savetxt("a_out.txt", a)
# save to file
b = np.loadtxt("a_out.txt")
# read from file
```

Array attributes

```
a = np.arange(10).reshape((2,5))

a.ndim     # 2 dimension
a.shape    # (2, 5) shape of array
a.size     # 10 # of elements
a.T        # transpose
a.dtype    # data type
```

Basic operations

```
a = np.arange(4)
# array([0, 1, 2, 3])

b = np.array([2, 3, 2, 4])

a * b  # array([ 0,  3,  4, 12])
b - a  # array([2, 2, 0, 1])

c = [2, 3, 4, 5]
a * c  # array([ 0,  3,  8, 15])
```

Vector operations

- inner product
- outer product
- dot product (matrix multiplication)

```
# note: numpy automatically converts lists
u = [1, 2, 3]
v = [1, 1, 1]

np.inner(u, v)
# 6
np.outer(u, v)
# array([[1, 1, 1],
#        [2, 2, 2],
#        [3, 3, 3]])
np.dot(u, v)
# 6
```

Matrix operations

```
A = np.ones((3, 2))
# array([[ 1.,  1.],
#        [ 1.,  1.],
#        [ 1.,  1.]])
A.T
# array([[ 1.,  1.,  1.],
#        [ 1.,  1.,  1.]])

B = np.ones((2, 3))
# array([[ 1.,  1.,  1.],
#        [ 1.,  1.,  1.]])
```

```
np.dot(A, B)
# array([[ 2.,  2.,  2.],
#        [ 2.,  2.,  2.],
#        [ 2.,  2.,  2.]])

np.dot(B, A)
# array([[ 3.,  3.],
#        [ 3.,  3.]])
```

Operations along axes

```
a = np.random.random((2,3))
# array([[ 0.9190687 ,  0.36497813,  0.75644216],
#        [ 0.91938241,  0.08599547,  0.49544003]])
a.sum()
# 3.5413068994445549
a.sum(axis=0)  # column sum
# array([ 1.83845111,  0.4509736 ,  1.25188219])
a.cumsum()
# array([ 0.9190687 ,  1.28404683,  2.04048899,  2.9598714 ,
#         3.04586687,  3.5413069 ])
a.cumsum(axis=1)  # cumulative row sum
# array([[ 0.9190687 ,  1.28404683,  2.04048899],
#        [ 0.91938241,  1.00537788,  1.50081791]])
a.min()
# 0.085995469040367
a.max(axis=0)
# array([ 0.91938241,  0.36497813,  0.75644216])
```

Slicing arrays

```
a = np.random.random((4,5))

a[2, :]
# third row, all columns
a[1:3]
# 2nd, 3rd row, all columns
a[:, 2:4]
# all rows, columns 3 and 4
```

Matrix operations

```
import numpy.linalg
```

`eye(3)`	Identity matrix
`trace(A)`	Trace
`column_stack((A,B))`	Stack column wise
`row_stack((A,B,A))`	Stack row wise

Linear algebra

```
import numpy.linalg
```

`qr`	Computes the QR decomposition
`cholesky`	Computes the Cholesky decomposition
`inv(A)`	Inverse
`solve(A,b)`	Solves $Ax = b$ for A full rank
`lstsq(A,b)`	Solves $\arg\min_x \|Ax - b\|_2$
`eig(A)`	Eigenvalue decomposition
`eig(A)`	Eigenvalue decomposition for symmetric or hermitian
`eigvals(A)`	Computes eigenvalues.
`svd(A, full)`	Singular value decomposition
`pinv(A)`	Computes pseudo-inverse of A

Fourier transform

`import numpy.fft`

- `fft` 1-dimensional DFT
- `fft2` 2-dimensional DFT
- `fftn` N-dimensional DFT
- `ifft` 1-dimensional inverse DFT (etc.)
- `rfft` Real DFT (1-dim)
- `ifft` Imaginary DFT (1-dim)

Random sampling

```
import numpy.random
```

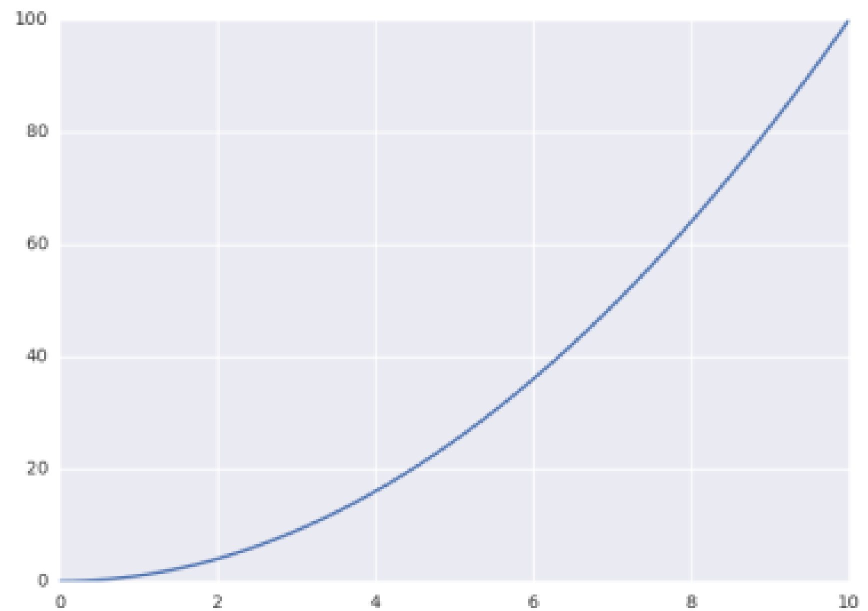
`rand(d0,d1,...,dn)`	Random values in a given shape
`randn(d0, d1, ...,dn)`	Random standard normal
`randint(lo, hi, size)`	Random integers [lo, hi)
`choice(a, size, repl, p)`	Sample from a
`shuffle(a)`	Permutation (in-place)
`permutation(a)`	Permutation (new array)

Matplotlib

- What is Matplotlib?
  - Plotting library for Python
  - Works well with Numpy
  - Syntax similar to Matlab

```python
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

x = np.linspace(0, 10, 1000)
y = np.power(x, 2)
plt.plot(x, y)
plt.show()
```



What is SciPy?

- SciPy is a library of algorithms and mathematical tools built to work with NumPy arrays.
  - linear algebra - scipy.linalg
  - statistics - scipy.stats
  - optimization - scipy.optimize
  - sparse matrices - scipy.sparse
  - signal processing – scipy.signal
  - **image processing: skimage**
    - pip install -U scikit-image
    - http://www.scipy-lectures.org/packages/scikit-image/index.html

Thank you!