Universität Potsdam
Institut für Informatik
Lehrstuhl Maschinelles Lernen

Introduction to Python

Gerrit Gruben

Plan for this lecture

- Introduction to Python
- Main goal is to present you a subset of the language and libraries to make you able to tackle Machine Learning challenges with Python.

Overview

- What is Python?
  - Python is an open general purpose language that is widely used in scientific computing and machine learning.
  - Rich ecosystem of libraries for scientific computation. NumPy for linear algebra, scikit-learn for general Machine Learning, Apache Spark for distributed ML and so on.

Python

- Python is dynamically typed, that means that the type of an expression is unknown before evaluation time. (but there are types!).
- Supports object-orientation and functional programming.
- We use Python 3 in the lecture.

Python Basics I

- Hello World:

```
print("Hello, World!")
```

- Variables:

```
x = 5
print(x)
print(type(x))
print("x = " + x)              # does not work
print("x = " + str(x))
```

- Arithmetic:

```
x = 0.5
y = 3
print(y**2 + x)              # y² + x
print(y / x)
z = 5
print(y / z)
print(float(y) / z)
print(int(x))
```

Python Basics II

- Boolean algebra:

```
winter = True
rain = False
snow = winter and rain
print(snow)
summer = not winter
print(summer)
bad_weather = winter or rain
print(bad_weather)
```

- Comparison operators:

```
print(5 == 3)                # note that = is an assignment
print(3 < 4)
print(2+2 == 5 and True)
print(4 % 3 == 0)            # a % b is remainder of integer
                             # division of a by b
```

Python Basics III

- Functions (notice the indentation!):

```
def square_plus(x, y):
  print("square with x= " + str(x) + " evaluated")
  return x*x + y
print(square_plus(3, 1))
```

- Call-by-???:

```
def set_to(x, value):
  print("x set to "+ str(value))
  x = value

y = 5
set_to(y, square_plus(2, 0)) # what happens in
                             # which order here
                             # did the original y change?
print y
```

Python Basics IV

- If (run specific code only if a condition is met):

```
def abs(x):
  if x < 0:
    return -x
  return x
print(abs(3), abs(-5))
```

- While (run code while some condition is met prior to each run)

```
i = 4
while i >= 0:
  print(i)
  i -= 1              # equal to i = i - 1
```

- For (run code for each object in an ordered sequence as a parameter)

```
for i in range(5):
  print(i)
for c in "WOW!":
  print(c)
```

Python Basics V

- Recursion (calling itself-itself-itself-…):

```
def factorial(n):
    if n == 0:
     return 1
   else:
     return n * factorial(n-1)

print(factorial(42))
```

Python Basics VI

- A test: find a good name for the following function

```
def what_am_i(n):
  i = 0

  while i < n:
    s = ""

    for j in range(n):
      if j == i or j == n-i-1:
        s += "*"
      else:
        s += " "
    print(s)
    i += 1
```

Python Basics VII

- Modules: Every file (or directory with _init_.py)

```
import math

print(math.sin(3))
print(math.factorial(10))

from math import sin, cos, exp
print(sin(3)**2 + cos(3)**2)
print(exp(1))
```

- Other module: random numbers

```
from random import *

print(random())              # 0 to 1 uniform
print(randrange(10))         # integer 0, 1, .., 9
print(uniform(-0.5, 0.5))
print(gauss(0, 1.0))         # normal distribution
```

- Standard modules:

```
collections, string, itertools, os, sys
```

Python Basics VIII (Data Structures I)

- Lists:

```
xs = [1, 2, 3, 4]
print(xs[0])
print(xs)
print(len(xs), sum(xs))
print([1, 2] + [3]*2 + [])
print("ab" * 3)
```

- Mutability of lists:

```
some_objects = []
some_objects.append("a")
some_objects.append(True)
some_objects.append(3)

print(some_objects[0])
print(some_objects)

del some_objects[0]
some_objects.remove(3)
```

Python Basics IX (Data Structures II)

- Slicing

```
nums = range(20)
print(nums[1:10])
print(nums[:10])
print(nums[5:])
print(nums[:])
print(nums[10::-1])
print(nums[:10:-1])
print(nums[3:15:4])
```

- List comprehensions

```
squares = [x**2 for x in range(10)]
print(squares)

pythagorean_triples = [(x, y, z) for x in range(1, 10)
for y in range(1, 10)
for z in range(10) if x**2 + y**2 = z**2 and x > y]
print(pythagorean_triples)
```

Python Basics X (Data Structures III)

- Dictionaries (hash maps):

```
dictionary = {"Eins": 1, "Zwei": 2, "Drei": 3}
print("Eins" in dictionary)
print 1 in dictionary
del dictionary["Eins"]

for key, value in dictionary.items():
  print("{}: {}".format(key, value))
```

- Lambda expressions

```
squaring = lambda x: x**2
print(squaring(3))
```

When you are stuck

- `help` opens documentation.
- `doc(obj)` or `obj?` for any object `obj` (commands, classes, modules)
- `who, whos`: lists all currently available identifiers, latter with more detail.
- `del x`: deletes x from memory.
- `clear`: clears output if you run Python in a terminal.

NumPy I

Start with `import numpy as np`

- Input of numbers:

```
>> a=2
2
>> a = np.sqrt(-16 + 0j)
4j
```

- With print explicit display of value:

```
>> print(a)
4j
```

- Or simply writing the name as last expression:

```
>> a
4j
```

NumPy II

- Defining a vector:

```
>> b = np.ndarray([2, 4, 6, 8])
[2 4 6 8]
```

This is a vector of length 4 (implicitly row vector)

```
>> b2 = b.reshape(4, 1)
>> print b.dot(b2)
array([120])
```

The data lies flat (i.e. sequentially) in memory, shape returns logical structure

```
>> print(b.shape, b2.shape)
(4,) (4, 1)
```

Shapes can be in any higher dimensions, ndarrays are in fact tensors.

NumPy III

- Generate c equidistant points from interval [a, b]:

```
>> b2 = np.linspace(1, 3, 5)
array([ 1., 1.5, 2., 2.5, 3. ])
```

- Generate range as a vector:

```
>> b3 = np.arange(0, 10, 2)
array([0, 2, 4, 6, 8])
```

NumPy IV

- Input of Matrices:

```
>> A = np.ndarray(np.mat(`[1 2 3; 4 5 6; 7 8 0]`))
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 0]])
```

  This results in a 3x3 matrix.

- Transpose:

```
>> A2 = A.T
array([[1, 4, 7],
       [2, 5, 8],
       [3, 6, 0]])
```

NumPy V

- Linear Indexing:

```
>> A[0]
array([1, 2, 3])
```

- Indexing over row and column:

```
>> A[1, 2]
```
returns 6, zero-based (row, column)

- Indexing via lists and slicing:

```
>> A([0,2],1)
```
returns [2, 8]

```
>> A[2,:]
```
returns 3rd row as slice

```
>> A[:,2]
```
returns 3rd column as slice

NumPy VI

- Change values via assignment:

```
>> A[2,2] = 9
A =
     1     2     3
     4     5     6
     7     8     9
```

- Matrix shape can be adjusted by reshape, but should not. Create new matrices by operators and creators.
- Information about matrices

`A.shape` Dimensions, returns (3, 3) here

`A.dtype` Kind of scalars the matrix contains, i.e. int64, float64

NumPy VII

- Commands to create matrices:

`np.zeros((n,m))`	nxm matrix with only zeros
`np.ones((n,m))`	nxm matrix with only ones
`np.full((n,m),c)`	nxm matrix with only c
`np.eye(n)`	nxn identity matrix

- Random sampling (more at SciPy docs)

```
from np.random import rand, randn
```

`rand(n,m)`	nxm matrix with uniform picked entries in the half-open unit interval [0, 1)
`randn(n,m)`	nxm matrix with normally distributed entries (zero mean, unit std)

NumPy VIII

- Some constants

`np.pi`	3.14159...
`0.+1j`	imaginary unit
`np.inf`	infinity
`np.nan`	“not a number“

NumPy IX

- Matrix operators:

`+`	addition
`-`	subtraction
`np.dot`	matrix multiplication
`^`	matrix exponentiation
`np.linalg.solve`	left division
`.T`	transpose
`.H`	complex-conjugated transpose

- Element-wise operators:

`*`	element-wise multiplication
`**`	element-wise exponentiation
`/`	element-wise division

NumPy X (Examples)

```
>> x = np.ndarray([-1, 0, 2])
   array([-1, 0, 2])

>> y = x - 1
   array([-2, -1, 1])

>> x.T.dot(y)
   4

>> x.dot(y.T)
   [adjusted output]
        2       1      -1
        0       0       0
       -4      -2       2

>> y.dot(x.T)
   [adjusted output]
        2       0      -4
        1       0      -2
       -1       0       2

>> np.pi * x
   array([-3.1416, 0., 6.2832])
```

Sources

- https://continuum.io – Anaconda distribution, easy to use installation of Python. Works well under Windows. This is also installed for you on the computer lab's Linux systems.
- http://learnpythonthehardway.org – A gentle introduction to Python as a general-purpose language.
- https://www.edx.org – Decent (and free) online classes for Python.
  - 6.00.2x: Python intro with scientific/statistical approach. If you lack CS fundamentals start with 6.00.1x.
  - CS190-1x: Large scale ML with Python and Spark. Labs very cool (e.g. visualization of neuroimage data of Jellyfishes).

More Sources

- https://github.com/amueller – Wonderful collection of tutorials for ML with Python with notebooks, you can find accompanying videos often.
- https://github.com/parallel_ml_tutorial -- Parallel ML with Python. Useful for quicker prototyping.