Pattern Recognition Python Intro

Fabian Wolf   Gernot A. Fink

Computer Science 12,
Pattern Recognition in Embedded Systems
TU Dortmund University

Summer Term 2019
Organization

The tutorials will be given in two groups:

**Group A** 15.07. - 19.07.2019, 9 - 17 Uhr

**Group B** 05.08. - 09.08.2019, 9 - 17 Uhr

**Room: OH16 U09**

- A tutor will be in the pool for approx. 4h/day.
- The exercises will follow the content of the lecture.
- They are very important to get additional and practical insights for the methods presented in the lecture
Welcome to the Pattern Recognition Python Intro

In this intro, you shall familiarize with the basic structures and functions of Python, NumPy/SciPy, matplotlib and the open computer vision library (OpenCV). This intro should help you to get started more easily with the tasks in the tutorials. More detailed information is linked on the website.
https://docs.python.org/3/tutorial/
Python

Python...

▶ is an easy and powerful programming language.
▶ has efficient high-level data structures.
▶ has a simple but effective approach to OOP.
▶ is an ideal language for scripting and rapid application development.
▶ its interpreter and standard library are freely available.
Python

Python...

- enables programs to be written compactly and readably.
- Programs written in python are, in general, much shorter than C, C++ or Java ones.
- is an interpreted language because no compilation and linking is necessary.
The interpreter...

- can be used interactively
- helps experimenting with features of the language.
  - to write throw-away programs.
  - to test functions during bottom-up program development.
- is also a handy desk calculator.
- Invoking the Interpreter

```python
$ python
Python 2.7.9 (default, Jun 29 2016, 13:08:31)
[GCC 4.9.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> 2 + 2
4
>>> 48 - 9 * 2
30
>>> 45 / 3 * 5
75
>>> 8 / 5
1
>>> 8.0 / 5
1.6
>>> 17 // 3
5
>>> 17 / 3
5
>>> 
```
General structure of a python script

'''  
Created on Jun 19, 2017  
@author: fmoya  
'''  

# Import packages  
import numpy as np  
import math  

def function_name(args, args2 = 0):  
    # Definition of functions  
    #@param args: arguments of the function  

    return  

def main(args):  
    #@param args: arguments of the function  

    return  

if __name__ == '__main__':  
    main()  
    # Invoking a function
Data types

Basic ones Variable typing is dynamic

- Int
- Float
- String: strings are immutable.

Data structures for sequences

- Tuples: consisting of a number of values separated by commas between round brackets. Tuples are immutable
  
  \[ \text{tup\_var} = (1, \ 1.0, \ '1.0') \]

- Lists: consisting of a number of values separated by commas in squared brackets.
  
  \[ \text{list\_var} = [1, \ 4, \ 9, \ 16, \ 25] \]
Data structures for sequences

Tuples are immutable, so they are, in general, a good container for passing arguments to/from functions.

For example

def sum_plus_one(sequence):
    try:
        sequence[0] += 1
        sum_of_sequence = sum(sequence)
    except TypeError:
        sum_of_sequence = sum(sequence)
        sum_of_sequence += 1
    return sum_of_sequence

.
.
.
test_list = [1, 2, 3]
test_tuple = (1, 2, 3)
print sum_plus_one(test_list) ---> 7
print sum_plus_one(test_tuple) ---> 7

print test_list ---> [2, 2, 3]
print test_tuple ---> (1, 2, 3)
if Statements

```python
variable = (1, 1.0, '1.0')
var21 = 21
var42 = 42
if var21 in variable:
    print var21, ' is in variable'
elif var42 in variable:
    print var42, ' is in variable'
else:
    print var21, ' and ', var42, ' arent in variable'
return
print '21 and 42 arent in variable'
```

The condition are boolean expressions using and, or, in / not in.
for Statements

The for statement differs from C, C++ (iterating over arithmetic progression of numbers or defining an iteration step and a halting condition). It is used to iterate over the elements of a sequence (such as a string, tuple or list) or other iterable object.

```python
sequence = []
# It could be a list, tuple or array
for element in sequence:
suite ...
break # for breaking the
# smallest enclosing for loop
continue # continues with the next
# iteration of the loop

words = ['cat', 'window', 'defenestrate']
for w in word:
    print w
____________
cat
window
defenestrate
```
List comprehensions are an efficient and powerful method for generating lists.

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

[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
Some useful functions

- range() generates arithmetic progressions-

```python
range(start, stop[, step])
```

```python
range(10)
---------------
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- enumerate() helps making clearer the loops. It allows us to loop over something and have an automatic counter.

```python
enumerate(sequence)
```

```python
sequence = ['dog', 'cat', 'elephant']
for n, v in enumerate(sequence):
    print "c, v"
-----------------
0 dog
1 cat
2 elephant
```
Dictionaries

A dictionary is a hashmap that associates each object with a key.

```
dict_var = key1: object, key2: object
```

dic_var = {‘anna’: 3098, ‘marco’: 4139}
dic_var[‘guido’] = 4127
print dic_var
print list(tel.keys())
print sorted(tel.keys())

------------------
{‘marco’: 4139, ‘anna’: 3098, ‘guido’: 4127}
Classes

A dictionary is a hashmap that associates each object with a key.

class Name_of_the_class(object):
    def __init__(self, arguments):
        """Constructor""
    def function(self, arguments):
        code

class DerivedClassName(BaseClassName):
    def __init__(self, arguments):
        """Constructor""
    def function(self, arguments):
        code
SciPy / NumPy

- To work with large amounts of data, the iterative approaches we have seen so far do not work well in Python. An alternative is the NumPy / SciPy library, which allows to execute numerical operations on vectors / matrices very fast.
- SciPy contains modules for optimization, linear algebra, integration, interpolation, FFT, signal and image processing, ODE solvers and other tasks common in science and engineering.

Hence, the idea is to execute matrix computations instead of computing mathematical operations iteratively.

For importing NumPy:

```python
import numpy as np
```

The basic data structure is the ndarray. A new ndarray can be created, for example from a python list:

```python
var_arr = np.array(sequence_list)
```
SciPy / NumPy

An empty array can be created by the zeros function.

```python
zeros_arr = np.zeros((3, 3))
print zeros_arr
```

```
[[ 0.  0.  0.]
 [ 0.  0.  0.]
 [ 0.  0.  0.]]
```

Having a look accessing ndarrays. First, one creates a 3x3 matrix containing the numbers from 0 to 8. Here, one employs the numpy range function and reshape the 1D array into 2D.

```python
seq_arr = np.arange(9).reshape(3, 3)
print "seq_arr dimensions:" , seq_arr.ndim
print "seq_arr shape:" , seq_arr.shape
print "seq_arr content:"
print seq_arr
```

```
seq_arr dimensions: 2
seq_arr shape: (3, 3)
seq_arr content:
[[0 1 2]
 [3 4 5]
 [6 7 8]]
```
NumPy: Operations

▶ Slicing

```python
print seq_arr ---->[[0 1 2]
            [3 4 5]
            [6 7 8]]
print seq_arr[0:2,0:2] ---->[[0 1]
                          [3 4]]
print seq_arr[-2:,-2:] ---->[[4 5]
                          [7 8]]
print seq_arr[1,:] ---->[3 4 5]
```

▶ Mathematical ops:

```python
print "Element-wise sum ", seq_arr[0] + seq_arr[1]
print "Element-wise product", seq_arr[0] * seq_arr[1]
```

```
Element-wise sum [3 5 7]
Element-wise product [ 0 4 10]
```

▶ Important methods:

```python
print "Max per row: ", np.amax(seq_arr, axis=1)
print "ArgMax per row: ", np.argmax(seq_arr, axis=1)
```

```
Max per row: [2 5 8]
ArgMax per row: [2 2 2]
```
Matplotlib

For data visualization, one uses the Matplotlib. matplotlib.pyplot is a collection of command style functions that make matplotlib work like MATLAB
https://matplotlib.org/users/pyplot_tutorial.html

```python
import matplotlib.pyplot as plt
plt.plot([1,2,3,4])
plt.ylabel('some numbers')
plt.show()
```
Example of bar plot

- Creates a figure object.

```python
fig = plt.figure()
```

- add subplot returns an axis object in the figure, where $r$ and $c$ define a grid of axis with $r$ rows and $c$ columns. The axis object at index $i$ is returned. The index starts at one and is first incremented along the rows.

```python
ax_obj = fig.add_subplot(111)
```

- Creates a bar plot in the axis object, given the $x$ and $y$ values. Furthermore, you can specify optional parameters such as width, alignment and alpha.

```python
ax_obj.bar(x_values, y_values, width = 0.9, align = 'center', alpha = 0.4)
```

- Finally, set some nice labels.

```python
ax_obj.set_xticks(np.linspace(0, len(y_values), len(x_values)))
```
Matplotlib

One shows the plot.

```python
plt.show()
```
OpenCV

So far the whole introduction did not show you a single image. In this last task we are going to change that. As you can see in the import section, we also imported cv2. Which symbolizes version 2 of the open computer vision library (OpenCV). It offers a variety of high level function calls in python:

openCV-python-tutroals.readthedocs.org/en/latest/

- Read an image

```python
import cv2

# Read an image
img = cv2.imread("../../data/double_rainbow.jpg")

print(type(img))
print(img.shape)
```

```
<type 'numpy.ndarray'>
(508, 1000, 3)
```
OpenCV

- Import package

```python
import cv2
```

- To display the image

```python
cv2.imshow('Double rainbow', img)
```

- Image is not shown. It has been, but only for a very short time as the imshow call is supposed to be non blocking. To make it blocking, one can use the waitKey method to wait for keyboard input.

```python
cv2.waitKey(0)
```
OpenCV

- Slicing:

```python
import cv2

# Slicing:
img = cv2.imread('image.jpg')

# Display the sliced image
cv2.imshow('Doublerainbowslice', img[:, 300:300])
```

- Play with the channels: BGR

```python
# Play with the channels: BGR
img_2 = img
img_2[:,:,0] = 0
img_2[:,:,1] = 0

# Display the filtered image
cv2.imshow('Double rainbow red', img_2)
```