text_mining_tutorial_01

October 17, 2019

0.1 Outline:

1. Lab class organization
2. Intorduction to Python
    1. Installation
    2. Basic operations
    3. Important libraries

0.2 Lab class organization:

- Lab announcements can be found on the course page and your contact is Milad Alshomary (milad.alshomary@uni-paderborn.de)
- 6 assignments in total (one assignment sheet every two weeks) covering a handful of text mining tasks.
- Assignment submissions in groups of maximum 3 students:
    - Submit your assignments via email to milad.alshomary@uni-paderborn.de.

      –tm-lab.zip |____ written_part.pdf ## Your names and student numbers should be included in this file also |____ programming_part.ipynb
- Assignments will be presented in the Lab right after the announcement date and can be found under the course page.
- Solutions and corrections will be discussed in the Lab session right after the submission date.

0.2.1 Programming Language : Python

- **Easy to learn**: Popular programming language for indroductory courses in computer science.
- **Productivity**: A lot done in a few lines of Python code
- **Interactive** programming language.
- **Libraries for Natural language processing (NLP) tasks**: tokenization, stemming, tagging, parsing, and text classification.

0.3 Introduction to python

Installing Python:

- On linux, most likely you have it. Otherwise:

  ```
  xx install python3 # xx is (apt-get, dnf, ...) based on you linux distribution
  ```

- On Windows, download the installer
- Test if python working:

  ```
  python3 --version
  ```

- Doesn't work? google it or simply ask us for help!

Installing Jupyter:

- An open-source web application for sharing documents that contain live code, visualizations and text.
- Assignments are given as python notebooks templates.
- Install Jupyter via pip command:

  ```
  pip install jupyter
  ```

- Run Jupyter:

  ```
  jupyter notebook
  ```

- Access the notebook on under localhost:8888

0.3.1 Basic operations:

```
In [1]: print("hello students!!")
```

```
hello students!!
```

```
In [2]: import math

        # square root of a number
        sq = math.sqrt(9)
        #print
        print(sq)
```

```
3.0
```

```
In [3]: print(dir(math))
```

```
['__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh',
```

```
In [4]: math.log(1)
```

```
Out[4]: 0.0
```

```
In [5]: help(math.log)
```

```
Help on built-in function log in module math:

log(...)
    log(x[, base])

    Return the logarithm of x to the given base.
    If the base not specified, returns the natural logarithm (base e) of x.
```

0.3.2 Variables

- Declaring variables
- Python is typeless programming lagnuage

```
In [2]: var1 = "Hello Python world!"
        var2 = 10
        var3 = 6.8
```

```
In [3]: type(var1)
```

```
Out[3]: str
```

```
In [4]: x = var2 + var3

        print(x)
```

```
16.8
```

0.3.3 Operators

```
In [9]: print(1 + 2, 1 - 2, 1 * 2, 1 / 2)
```

```
3 -1 2 0.5
```

```
In [10]: print(True and False, False and False, True or False, not False)
```

```
False False True True
```

0.3.4 Strings

- Define a string.
- A string is sequence of characters.
- Operations on strings (concatenate, contains, title, ...)

```
In [1]: s1 = "first name"
        s2 = "last name"

        # concatenation of strings
        print( s1 + " - " + s2)

        # String is an array of characters..
        print(s1[0])

        # Length of a string
        print(len(s1))

        # Return first occurance in a string
        print(s2.find('Hello'))

        #Return the index of first occurance of letter 'e'
        print(s2.index('e'))
```

```
first name - last name
f
10
-1
8
```

```
In [12]: print(dir(s1))
```

```
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__forn
```

0.3.5 Lists and Tuples:

- Constructing a list.
- Operations on lists (concatenation, find element, length, sort, reverse, slicing ...)
- Tuples are basically lists that can never be changed.
- Loops over lists & Tuples

```
In [13]: students = ['tall student', 'smart student', 'funny student']


         print(type(students))

         # Length of the list
         print(len(students))

         #printing content of the list
         print(students)
```

```
<class 'list'>
3
['tall student', 'smart student', 'funny student']
```

```
In [16]: # Accessing elements in the list
         print(students[0])

         # Slicing a list..
         print(students[0:2])

         # Adding element to a list
         students.append('new student')
         print(students[-1])

         #Concatenating two lists
         l1 = [1, 2, 3]
         l2 = [4,5]

         print(l1 + l2)
```

```
tall student
['tall student', 'smart student']
new student
[1, 2, 3, 4, 5]
```

```
In [17]: print(dir(l1))
```

```
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '
```

Tuples: The immutable lists.

```
In [9]: tuple1 = ('RED', 'GREEN', 'BLUE')

        print(tuple1[0])

        #changing element of tuple is not allowed because its immutable object
        tuple1[0] = 'R'
```

```
RED
```

```
        ---------------------------------------------------------------------------

        TypeError                                 Traceback (most recent call last)

        <ipython-input-9-2db7eb818fd1> in <module>()
          4
          5 #changing element of tuple is not allowed because its immutable object
    ----> 6 tuple1[0] = 'R'


        TypeError: 'tuple' object does not support item assignment
```

0.3.6 Dictionaries

- Defining dictionaries.
- Operations on dictionaries.

```
In [16]: params = {"parameter1" : 1.0,
                   "parameter2" : 2.0,
                   "parameter3" : 3.0,}

         print(type(params))
         print(params)
```

```
<class 'dict'>
{'parameter1': 1.0, 'parameter2': 2.0, 'parameter3': 3.0}
```

```
In [17]: params["parameter1"] = "A"
         params["parameter2"] = "B"

         # add a new entry
         params["parameter4"] = "D"

         print("parameter1 = " + str(params["parameter1"]))
         print("parameter2 = " + str(params["parameter2"]))
         print("parameter3 = " + str(params["parameter3"]))
         print("parameter4 = " + str(params["parameter4"]))
```

```
parameter1 = A
parameter2 = B
parameter3 = 3.0
parameter4 = D
```

0.3.7 Control Flow

- Using while and if statments

```
In [18]: statement1 = False
         statement2 = False

         if statement1:
             print("statement1 is True")
         elif statement2:
             print("statement2 is True")
         else:
             print("statement1 and statement2 are False")
```

```
statement1 and statement2 are False
```

```
In [20]: mylist = ["scientific", "computing", "with", "python"]

         for word in mylist:
             print(word)
```

```
scientific
computing
with
python
```

```
In [21]: i = 0
         while i < 5:
             print(i)
             i = i + 1
         print("done")
```

```
0
1
2
3
4
done
```

0.3.8 Functions:

- Defining a function..
- Calling a function..
- Anonymous (lambda) functions for functional programming..

```
In [22]: # include a docstring
         def func(s):
             """
             Print a string 's' and tell how many characters it has
             """

             print(s + " has " + str(len(s)) + " characters")
```

```
In [23]: help(func)
```

```
Help on function func in module __main__:

func(s)
    Print a string 's' and tell how many characters it has
```

```
In [24]: func('hello!')
```

```
hello! has 6 characters
```

0.3.9 Lambda & Map and Filter:

- **Lambda** operator or lambda function is used for creating small, one-time and anonymous function objects in Python.
- **Map function** expects two arguments, a `function_object` and an Iterable object. It executes the `function_object` for each element in the iterable object and returns a new Iterable object.
- **Filter function** expects two arguments, a `function_object` that returns a boolean value and an Iterable object. The `function_object` is called for each element of the Iterable and only those elements where the `function_object` resulted true are returned in a new Iterable.

Lambda example:

```
In [3]: # Defining anonymous functions
        # Functions are like variables can be passed to other functions..
        def get_square_func():
            #Defined an anonymous function and save it in variable f1
            f1 = lambda x: x**2
            return f1

        squre_fun = get_square_func()
        squre_fun(2)
```

```
Out[3]: 4
```

Map and Filter example:

```
In [3]: # List of names
        names = ['Simon', 'Max', 'Pascal']

        def filter_long_names(name):
            if len(name) > 3:
                return True
            else:
                return False
        def add_affix(name):
            return 'Mr. ' + name

        # Keep only names with three characters
        filtered_names = filter(filter_long_names, names)
        # Add affix to all names
        filtered_names = map(add_affix, filtered_names)

        print(list(filtered_names))
```

```
['Mr. Simon', 'Mr. Pascal']
```

Same task with few lines of code:

```
In [6]: # Keep only names with three characters
        filtered_names = filter(lambda x: len(x) > 3, names)
        # Add affix to all names
        filtered_names = map(lambda x: 'Mr. ' + x, filtered_names)
        # Printing results
        print(list(filtered_names))
```

```
['Mr. Simon', 'Mr. Pascal']
```

0.3.10 Classes:

- Defining a class
- Initializing an object from a class..
- calling a method on the class

```
In [44]: class Point:
             def __init__(self, x, y):
                 self.x = x
                 self.y = y

             def translate(self, dx, dy):
                 self.x += dx
                 self.y += dy

             def __str__(self):
                 return("Point at [%f, %f]" % (self.x, self.y))
```

```
In [45]: p1 = Point(0, 0.5)
         print(p1)
```

```
Point at [0.000000, 0.500000]
```

```
In [46]: p1.translate(0.25, 1.5)

         print(p1)
```

```
Point at [0.250000, 2.000000]
```

0.3.11 Exercise:

Given a list of tuples (city, temperature in celsius) `cities` we want to apply two operations:

- Transform the degrees from celsius to fahrenheit
- Remove cities that have fahrenheit more than 50

```
In [19]: cities = [('Berlin', 14), ('Leipzig', 8), ('Paderborn', 17)]

         # 1. Method one:

         def fahrenheit(city):
             city_name, temp = city
             temp_fahrenheit = ((float(9)/5)*temp + 32)
             return (city_name, temp_fahrenheit)

         def more_than_50(city):
             return city[1] >= 50

         cities = map(fahrenheit, cities)
         cities = filter(more_than_50, cities)
         print(list(cities))
```

```
[('Berlin', 57.2), ('Paderborn', 62.6)]
```

```
In [20]: cities = [('Berlin', 14), ('Leipzig', 8), ('Paderborn', 17)]

         # 2. Method two:

         #map degree from celcius to fahrenheit
         cities = map(lambda x: (x[0], (float(9)/5) * x[1] + 32), cities)
         #filter out cities with fahrenheit less than 50
         cities = filter(lambda x: x[1] >= 50, cities)
         print(list(cities))
```

```
[('Berlin', 57.2), ('Paderborn', 62.6)]
```

0.3.12 Important Libraries:

- Any library can be downloaded by using the pip tool:

```
pip install library-name    # library-name = matplotlib, numpy, nltk ....
```

- In this lecture we will be using the following libraries:
  - numpy
  - nltk
  - matplotlib

Numpy (http://www.numpy.org/)

- The core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays

```
In [2]: import numpy as np

        a = np.array([1, 2, 3])    # Create a rank 1 array
        print(a.shape)             # Prints "(3,)"

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

        #Slicing the matrix
        print(b[0,0]) # prints the element in row=0, column=0
        print(b[0:2, 0:2]) # slice the matrix by taking the first two columns, and two rows
        print(b[-1, 0:2]) # retrieve the first two columns of the last row
```

```
(3,)
(2, 3)
1
[[1 2]
 [4 5]]
[4 5]
```

```
In [3]: 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.]]"


        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
```

```
[[0. 0.]
 [0. 0.]]
[[1. 1.]]
[[1. 0.]
 [0. 1.]]
[[0.34226907 0.6347119 ]
 [0.03753865 0.71115787]]
```

```
In [4]: x = np.array([[1,2],[3,4]], dtype=np.float64)
        y = np.array([[5,6],[7,8]], dtype=np.float64)

        print(x + y)
        print('========')
        print(x - y)
        print('========')
        print(x * y) #Element wise multiplication
        print('========')
        print(x / y)
        print('========')
        print(np.sqrt(x))
        print('========')
        print(np.dot(x, y)) #Matrix multiplication
```

```
[[ 6.  8.]
 [10. 12.]]
========
[[-4. -4.]
 [-4. -4.]]
========
[[ 5. 12.]
 [21. 32.]]
========
[[0.2        0.33333333]
 [0.42857143 0.5       ]]
========
[[1.         1.41421356]
 [1.73205081 2.        ]]
========
[[19. 22.]
 [43. 50.]]
```

NLTK (https://www.nltk.org/)

- nltk is a leading platform for building Python programs to work with human language data.

  Tokenization:

```
In [5]: import nltk

        text = """ At eight o'clock on Thursday morning Arthur didn't feel very good."""
        tokens = nltk.word_tokenize(text)
        print('Tokens:' , tokens)

        text = """ Diabetes mellitus is a group of metabolic diseases characterized by high bl
        levels that result from defects in insulin secretion, or its action, or both.
        Diabetes mellitus, commonly referred to as diabetes (as it will be in this article)
        was first identified as a disease associated with "sweet urine," and excessive muscle
        """
        sents = nltk.sent_tokenize(text)
        print('Sentences:', sents)
```

```
Tokens: ['At', 'eight', "o'clock", 'on', 'Thursday', 'morning', 'Arthur', 'did', "n't", 'feel'
Sentences: [' Diabetes mellitus is a group of metabolic diseases characterized by high blood s
```

Text Corpora:

```
In [7]: from nltk.corpus import gutenberg
        #Downloading gutenburg dataset
        nltk.download('gutenberg')
```

```
[nltk_data] Downloading package gutenberg to
[nltk_data]     /home/miladalshomary/nltk_data...
[nltk_data]   Package gutenberg is already up-to-date!
```

```
Out[7]: True
```

```
In [8]: macbeth_sentences = gutenberg.sents('shakespeare-macbeth.txt')
        print('Sentences:', macbeth_sentences[0:10])
        print('======================')

        macbeth_words = gutenberg.words('shakespeare-macbeth.txt')
        print('Words:', macbeth_words[0:10])
        print('======================')
```

```
Sentences: [['[', 'The', 'Tragedie', 'of', 'Macbeth', 'by', 'William', 'Shakespeare', '1603',
======================
Words: ['[', 'The', 'Tragedie', 'of', 'Macbeth', 'by', 'William', 'Shakespeare', '1603', ']']
======================
```

Stemming:

```
In [9]: from nltk.stem import PorterStemmer
```

```
        ps = PorterStemmer()

        words = ["game","gaming","gamed","games"]

        print(list(map(lambda x: ps.stem(x), words)))

['game', 'game', 'game', 'game']
```

Part of Speech Tagging:

```
In [10]: from nltk import pos_tag

         document = """Today the Netherlands celebrates King\'s Day.
         To honor this tradition, the Dutch embassy in San Francisco invited me to'
         """
         sentences = nltk.sent_tokenize(document)
         print('Sentences:')
         print(sentences)
         data = list(map(lambda x: nltk.pos_tag(nltk.word_tokenize(x)), sentences))
         print('\nPOS tags:')
         print(data[0]) #printing the tokens of the first sentence and thier POS tags

Sentences:
["Today the Netherlands celebrates King's Day.", "To honor this tradition, the Dutch embassy i

POS tags:
[('Today', 'NN'), ('the', 'DT'), ('Netherlands', 'NNP'), ('celebrates', 'VBZ'), ('King', 'NNP'
```

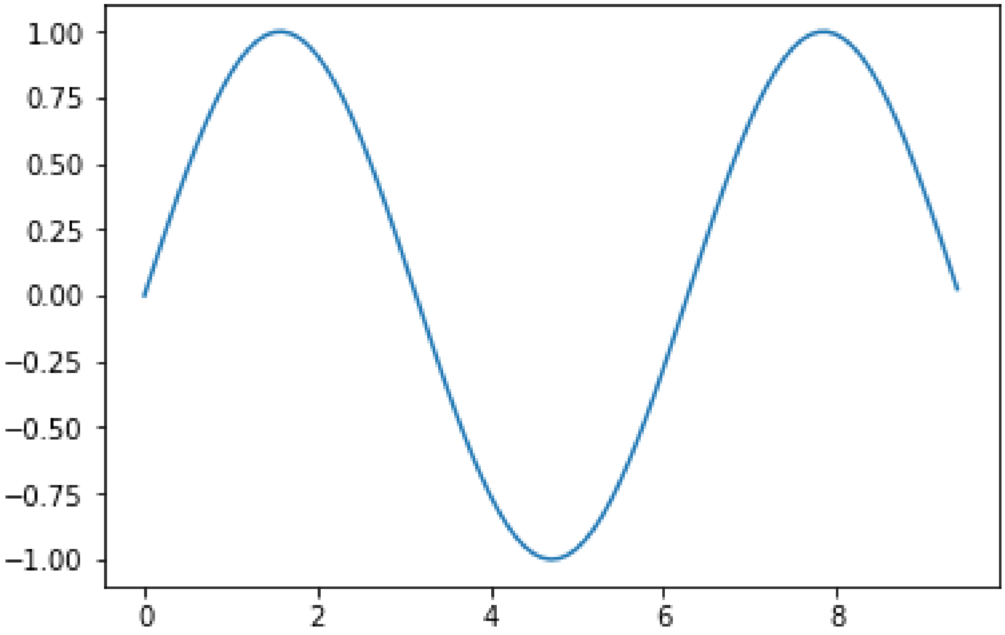
Matplotlib (https://matplotlib.org/)

- Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms

```
In [12]: import numpy as np
         import matplotlib.pyplot as plt

         # Compute the x and y coordinates for points on a sine curve
         x = np.arange(0, 3 * np.pi, 0.1)
         y = np.sin(x)

         # Plot the points using matplotlib
         plt.plot(x, y)
         plt.show()  # You must call plt.show() to make graphics appear.
```


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
2
4
6
8