prac1_solution

September 12, 2019

1 Introduction to Data Mining

This is the first tutorial - it provides a quick introduction to Python and IPython.

1.0.1 Python

Python is a programming language that has been growing in popularity in recent years. There are many reasons for this, but it mostly comes down to Python being easy to learn and use as well as the fact that Python has a very active community that develops amazing extensions to Python!

Python has become one of the most frequently used languages in the world of data science due to the ability to almost instantly apply it to a large number of data science problems. When asking companies in different industries and of various sizes what lanuage they would like their data scientists to know when coming in, they almost all agree that Python is the best choice.

Most of you have had experience in Java. Both share many characteristics given they are both programming languages and many structures and concepts will be familiar. However, there are differences. You may find you prefer python and it may allow you to be more productive, not just because of the provision of many libraries which implement Data Mining algorithms.

Some key differences between Python and Java are: - Python is dynamically typed while Java is statically typed, this has various implications including you never need to declare variables in Python - Python code is more concise (briefer) - Python is more compact

For example, the following program is in Java: public class HelloWorld { public static void main (String[] args) { System.out.println("Hellold!"); } }

In Python the equivalent program is print("Hello, world!") # Python version 3

1.0.2 Python? IPython? IPython notebooks? What is all of this?

What is all of this? - Python? - IPython? - IPython notebooks?

For you, these terms may be confusing? Don't worry its ok...

Python is a lanuage. A computer programming language. It is very popular in lots of industries. It has been around for over 30 years. Development of Python started in December 1989 by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands.

IPython (with an I) is an extension to Python. It was created by scientists for scientists. It is typically used in computer science, machine learning, and physics research. It simply adds some new features to Python. Writing code in either of these is generally done in a long text document with a ton of code that can be pretty duanting to a someone new to the world of programming.

IPython Notebooks (with an I, there is no such thing as a Python notebook) is what we are looking at right now. It is a web browser based way of writing Python code. One of the benefits

is that it allows you to write in plain text to create, what should feel like, a notebook. The closest analog here would be to relate an IPython notebook to a typical lab notebook kept by "traditional" researchers. Anyone coming from chemistry or biology will probably understand what I mean. We will be using IPython notebooks for the rest of the semester. A majority of your class notes will be presented in this format. This will allow us to both have a place for discussion and instruction, but with the added benefit of allowing us to play with data live!

The rest of this document will be broken into two main pieces: an introduction to IPython notebooks (how to use them) and then an introduction to the world of Python programming.

1.0.3 IPython Notebooks

IPython notebooks are made up of cells. There are two basics types of entries in an IPython notebook: text cells, and code cells.

You can edit a cell by double clicking on it. You can get it back to the display mode by pressing the "Run" button from the toolbar. Try it! To switch between text and code cells, just click a cell and go to "Cell > Cell Type" in the menu bar (or use the toolbar).

Text Cells Let's start a new cell and add a little bit more text.

You can do text formatting. For example, you can use asterisks or underscores to emphasize things. Double asterisks are used to make things bold.

If you know what LaTeX is, you can write directly in LaTeX by wrapping it in dollar signs, $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$. Learning LaTeX is great for typesetting math formulas.

Creating a bulleted list is pretty easy: - One - Two - Three

It is also very easy to make a numbered list: 1. One 1. Two 1. Three

For a numbered list, start with 1. followed by a space, then it starts numbering for you. Start each line with some number (any number) and a period, then a space. Tab to indent to get sub-numbering.

This covers almost all of the text formatting you will need to learn. If you are ever stuck, just Google "Markdown syntax" since the language the formatting is done in is called Markdown.

Code Cells Now, we will see a "code" cell. Here, we simply type any Python code and then click "Run". When we run a cell, the code in it is executed and remembered for as long as we keep this window open - this means if you create a variable in a code cell you can refer to the variable by its name in other subsequent cells. Code cells will always start with "In [ ]:". Code cells are the default for any new cell.

```
In [4]: x = 5 + 10

In [7]: print (x)

15
```

You can include more complex expressions or programs in a cell as well:

```
In [9]: x = 5 + 1
        print ("The value of x is " + str(x) + ".")
        print ("Output: Hi! This is a cell. Press the [U+25B6]| button above to run it")
```

```
The value of x is 6.
Output: Hi! This is a cell. Press the [U+25B6]| button above to run it
```

Instead of clicking "Play" button, you can run a cell with Ctrl + Enter or Shift + Enter. Experiment with both of those to see what the difference is.

1.0.4 Python

Let us now examine some of the key elements of Python

Variables and data types Variables are used to store data. This data can be of a variety of types. Integer numbers, floating (decimal numbers), lists, strings, etc. Let's take a look at some of these:

```
In [8]: some_integer = 5
        some_float = 7.1
        some_list = [1, 2, 3, 4]
        some_string = "Rob"
```

We can print out these variables.

```
In [9]: print (some_integer)
        print (some_float)
        print (some_list)
        print (some_string)
```

```
5
7.1
[1, 2, 3, 4]
Rob
```

What if I want to print some text and then some numbers? One easy way to do this is to realize that printing will always want string data. If you have data that is not a string (like an integer or float), you can convert it to a string.

print ("My integer was " + str(some_integer) + ".")

It is always a good practice to convert everything to a string when printing it out such as by using the function str().

Now we look at some basic maths.

```
In [12]: some_integer + some_float
```

```
Out[12]: 12.1
```

We can store this as a new variable to use later,

```
In [15]: my_sum = some_integer + some_float
```

```
In [17]: print (my_sum)
```

```
12.1
```

What about that list we had? What does that mean? A list is exactly what it sounds like. It's a way to keep a collection of things in order. We can check to see how long our list is,

```
In [19]: print( len(some_list) )

4
```

This looks good. Our list contained the numbers 1 through 4. What if we want a particular item from the list? How do we look at just the first item? To do a lookup, we use square brackets. Notice that when we created the list originally, we also used square brackets!

```
In [20]: print( some_list[1] )

2
```

That's the second item, not the first! In Python (and almost every other language), counting start at zero! To get the first item we should look in the 0th space,

```
In [22]: print( some_list[0] )

1
```

Adding things to the list is done using the append method,

```
In [23]: some_list.append(5)

In [24]: print (some_list)

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

```
In [ ]: # Play around here!
        # By the way, the pound (hash) symbol here is used to indicate a comment in code.
```

Functions We have already used these twice! Functions allow us to do predefined operations. Functions are usually some sensible English word ending in open-and-close parentheses. One example is the s

```
In [25]: str(5.124)

Out[25]: '5.124'
```

We also used the append() function to add stuff to a list.

If we knew we had to do some operation many times, and wanted to save a bit of time, we could define our own function. For example, consider having to calculate the area of a circle.

```
In [26]: def area_of_a_circle(radius):
             area = 3.14 * radius * radius
             return area

In [28]: circle_area = area_of_a_circle(5)
         print( circle_area )

78.5
```

This function was helpfully named "area_of_a_circle", it takes one argument that we will call radius. It then uses this radius to get the area and then returns it. Now, whenever I want to get the area of some circle, I simply call area_of_a_circle() and place the radius in the middle of the parentheses.

Python has many functions, but we will be writing our own very often.

Loops We will be doing a lot of repetative things in Python. This doesn't mean we need to do a ton of copy and pasting, though. We can use loops to make this easy. For example, if we wanted to square each

```
In [30]: for number in [1, 2, 3, 4, 5]:
             print (number * number)

1
4
9
16
25
```

The range function makes this even easier,

```
In [31]: for number in range(5):
             print (number * number)

0
1
4
9
16
```

Not exactly the same... the range function will start from 0 and go to the last number minus one. We can fix this by telling it to start at 1:

```
In [33]: for number in range(1, 6):
             print (number * number)
```

```
1
4
9
16
25
```

We aren't limited to this, let's bring in another list:

```
In [38]: names = ["Robert", "John", "Sarah", "Qian", "Ahmad"]
         ages = [26, 31, 29, 24, 30]

         for i in range(len(names)):
             print (str(names[i]) + " is " + str(ages[i]) + " years old.")
```

```
Robert is 26 years old.
John is 31 years old.
Sarah is 29 years old.
Qian is 24 years old.
Ahmad is 30 years old.
```

Conditionals Sometimes we want to check something before deciding what to do next. For example,

```
In [39]: def is_best_prof(name):
             if name == "Adam":
                 return "Yes!"
             else:
                 return "No!"

In [42]: print (is_best_prof("Adam"))
```

```
Yes!
```

```
In [44]: print (is_best_prof("John"))
```

```
No!
```

Packages Python has a ton of packages that make doing complicated stuff very easy. We won't discuss how to install packages, or give a detailed list of what packages exist, but we will give a brief description about how they are used. An easy way to think of why package are useful is by thinking: "Python packages give us access to MANY functions!"

In this class we will use four packages very frequently: pandas, sklearn, matplotlib, and numpy:

- pandas is a data manipulation package. It let's you store data in data frames. More on this next class.

- sklearn is a machine learning and data science package. It let's you do fairly complicated machine learning tasks, such as running regressions and building classification models with only a few lines of code!
- matplotlib let's you make nice looking plots.
- numpy (pronounced num-pie) is used for doing "math stuff" such as complex math operations (e.g., square roots, exponents, logs) and give you complex matrix operation abilities. If it's confusing as to why this is useful, don't worry. As we use them throughout the semester, their usefulness will become apparent.

To make the contents of a package useful, you need to import it:

```
In [48]: import pandas
         import sklearn
         import matplotlib
         import numpy
```

We can now use some package specific things. For example, numpy has a function called sqrt() which will give us the square root of a numpy. Since it is part of numpy, we need to tell Python that's where it is by using a dot.

```
In [49]: numpy.sqrt(25)
```

```
Out[49]: 5.0
```

You may have noticed that earlier, when we added stuff to our list, we used .append(). This is very similar! Here, we told Python that numpy had a function called sqrt() that we would like to use. Earlier, we told Python that our list (and all lists!) had a function called append() that we would like to use.

That's all we say about packages for now. Soon, we will be using packages in every class. With practice, you will understand why they are so great!

1.0.5 IPython Notebooks

IPython notebooks make writing Python code easy and neat.

Auto complete One of the most useful things about IPython notebook is its tab completion.

Try this: click just after numpy. in the cell below and press Tab several times, slowly to view candidate functions you might want to call

```
In [ ]: numpy.
```

Organisation We typically read IPython notebooks from top to bottom. This means that if a cell relies on a variable or function that was created earlier in the notebook, you must run the corresponding cell to make that information available! For example, if I set the variable age equal to 26 in the next cell,

```
In [53]: age = 26
```

but don't run it, it will not be available in the next cell:

```
In [55]: print ("I am " + str(age) + " years old!")

I am 26 years old!
```

Now, this does not mean you have to run everything from top to bottom. You could define age later in the notebook, and then scroll up and run any other cells that require it. However, this is bad practice. How is someone supposed to know to scroll down first!? The IPython notebook will always remember the cells in the order you run them. This means you can overwrite variables!

```
In [57]: gender = "male"

In [58]: print (gender)

male


In [61]: gender = "female"

In [62]: print (gender)

female
```

This can get confusing. Notice that the number in the "In [#]:" statement will always increase by one for every cell you run. This will make keeping track of everything a little easier. But it is recommended to always recommend organizing from top to bottom.

Saving There is an autosave feature and you can also use the save button in the menu or toolbar.

1.0.6 Further Information

The resources here are endless; but people learn things in different ways. Some people prefer books, others like being taught through lessons, and others just learn by doing. I, unfortunately, learn by doing which means that I'm not an expert on which resources are the best.

- icourse163.org has python courses.
- Codecademy's Python Course will give you a great foundation for Python.
- The online book, Diving into Python, has proven to be useful for many people.
- ...
- Personally, I try to think of a cool project I want to work on and I just try to do it. For example, trying to build a simple interactive text game can be fun. Since you are just jumping in and doing Python, this will lead to a ton of furious searching online. This is normal. This is actually how professional programmers work every day... Stackoverflow.com for instance has an answer to any programming question you can imagine!

1.0.7 Coding Questions

To master your new found knowledge of Python, you should try these hands-on examples.

1. Create a list of 5 fruits (make sure to include an apple).

```
In [17]: my_list = ["apple", "pear", "orange", "pinapple","mandarin"]
         my_list
```

```
Out[17]: ['apple', 'pear', 'orange', 'pinapple', 'mandarin']
```

2. Go through each fruit and check if it is an apple. If it is, print out "I found it!". If it's not an apple, do nothing.

```
In [18]: for  fruit in my_list :
             if (fruit == "apple"):
                 print ("found it!")
```

```
found it!
```

3. Add two new fruits to your list.

```
In [19]: my_list.append("bannana")
         my_list.append("kiwi")
         my_list
```

```
Out[19]: ['apple', 'pear', 'orange', 'pinapple', 'mandarin', 'bannana', 'kiwi']
```

4. Create a new empty list. Go through your list of fruits, and for each one, add an entry to the new list that tells us how many letters each fruit name is.

```
In [34]: for  fruit in my_list :
             print ("{:s} has {:d} letters".format(fruit,len(fruit)))
```

```
apple has 5 letters
pear has 4 letters
orange has 6 letters
pinapple has 8 letters
mandarin has 8 letters
bannana has 7 letters
kiwi has 4 letters
```

5. Make a function called half_squared that takes a list and returns a new list where each element of the original is squared and then divided in half.

```
In [35]: def half_squared(input_list):
             output_list = [] # What should we do?
             for element in input_list:
                 output_list.append((element*element)/2)
             return output_list
```

```
In [36]: ## test the function
         half_squared([3,3]) == [4.5,4.5]
```

```
Out[36]: True
```