CS447: Natural Language Processing Fall 2015

Homework 0

Handed Out: 8/25/2015 *Due: 10pm, 9/10/2015*

Please submit an archive of your solution (including code) on Compass by 10:00pm on the due date. Please document your code where necessary.

Note about Homework 0

This homework is intended as a warm-up exercise to get you thinking about language and to get you to be familiar with Python. If you don't have experience with Python, we strongly recommend completing the assignment in a timely fashion; attend TA office hours or submit questions via Piazza if you have difficulties. **This assignment with worth 2 points towards your homework grade for the course.** You will submit your work via Compass ("Course Content" →"Homeworks" →"Homework 0"); see Section 2.7 for more detailed instructions.

1 Question 1: Basic linguistic knowledge

I read a very interesting novel and a dull textbook over the summer break.

Parts of speech You probably remember from secondary school that words have parts of speech such as noun, pronoun, verb, adjective, adverb, preposition, article, and conjunction. What parts of speech do each of the words in the above sentence have? For how many of these words can you find a sentence where they have a different part of speech? For example, *walk* in *I walk* is a verb, but in *I took a walk* it is a noun.

2 Question 2: Python

2.1 Our goal

In order to learn about Python (and some basic things we can do with text), we will be writing a Python script that reads in a text file and analyzes the data in a number of ways. First, we want to know some statistics about our data, e.g. the number of words it contains, or how often each word occurs. We will introduce the bits of Python that you'll need to solve the exercises, but for more information you should read the Python Tutorial `http://docs.python.org/tutorial/`, esp. sections 1–7.

2.2 Running Python

You should be able to start Python from a shell, e.g:

```
[ramusa2@linux-a1 ~]$ python
Enthought Canopy Python 2.7.6 | 64-bit | (default, Jun  4 2014, 16:32:15)
[GCC 4.1.2 20080704 (Red Hat 4.1.2-52)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Because of inconsistencies between Python versions 2 and 3, we've opted to use version 2.7 for this course; this version is compatible with the Enthought Canopy Python distribution that we will use later in the semester.

Both Python 2.7 and Canopy are available through the module system on the EWS workstations. You are also welcome to install the software on your own machine. See the appendix at the end of this handout for instructions on using the EWS machines or installing Python and Canopy on your own machine.

If you can run Python, this will open a Python shell (exit with Control-D or type `exit()`). Python is an interpreted language, so we could type everything into its own shell. But because we want to write larger programs, we prefer to save our code to a file, e.g. `hw0.py`. Files that contain python code are also called modules.

2.3 Code and data for this assignment

If you downloaded this assignment from either Compass or the syllabus page on the course website, then the archive should include two stub files: `hw0_notNested.py` and `hw0_Nested.py`. Your task is to finish implementing the functions defined in these modules (see section 2.6) and evaluate them on the included data file `movies.txt`.

2.4 Reading in a text file

In our python module, we will define various functions, e.g. to read a text file into a data structure we can process:

```
# Read a text file into a corpus
def readFileToCorpus(f):
    """ Reads in the text file f which contains one sentence per line.
    """
    file = open(f, "r") # open the input file in read-only mode
    corpus = [] # this will become a list of sentences
    print "reading file ", f
    for line in file:
        sentence = line.split()  # split the line into a list of words
        corpus.extend(sentence)  # extend the current list of words with the words in the sentence
    return corpus
```

As you can see, Python looks very similar to C or Java. The syntax (e.g. for for-loops) is a little different though – there are no semicolons at the end of statements, or parentheses around blocks of code. Instead, Python identifies blocks of code by indentation. It is therefore highly recommended that you use an editor that has a Python mode to help you with correct formating of your code! Unlike C or Java, Python does not explicitly type function arguments (or return types).

We will be using the file `movies.txt`, which contains sentences from movie reviews, one line per sentence:

```
[ramusa2@linux-a1 ~]$ head movies.txt
plot : two teen couples go to a church party , drink and then drive .
they get into an accident .
one of the guys dies , but his girlfriend continues to see him in her life , and has nightmares .
what's the deal ?
```

As you can see, this file has been preprocessed already to normalize spelling and to split off punctuation.

In order to be able to run our module `hw0.py` as a python script, we also need to provide a main routine, which is done by including the following at the end of the file:

```
if __name__ == "__main__":
    movieCorpus = readFileToCorpus('movies.txt')
```

(Make sure that `movies.txt` is in the same directory as your script.)

Now we can run it directly from the terminal, except that it doesn't do very much yet:

```
[ramusa2@linux-a1 ~]$ python hw0.py
```

Before we extend this script, let us introduce some of Python's data structures:

2.5 Python data structures

2.5.1 Sequences: lists, tuples, strings

Lists, tuples and strings are all examples of Python's built-in sequence data types. As such, they all allow the operations given in table 1 (you won't need many of these for the homework).

`x in s`	True if an item of s is equal to x, else False
`x not in s`	False if an item of s is equal to x, else True
`s + t`	the concatenation of s and t
`s * n, n * s`	n shallow copies of s concatenated
`s[i]`	ith item of s, origin 0
`s[i:j]`	slice of s from i to j
`s[i:j:k]`	slice of s from i to j with step k
`len(s)`	length of s
`min(s)`	smallest item of s
`max(s)`	largest item of s

Table 1: Common sequence operations: `s` is a sequence (e.g. a list, tuple, or string)

Strings String literals can be enclosed in single or double quotes. There are also some escape sequences such as `\n` (ASCII newline) and `\t` (ASCII tab).

```
>>> string1 = "this is a string"
>>> string2 = "this is also a string\n"
>>> string3 = 'this is another string\n'
>>> print string1
this is a string
>>> print string2
this is also a string

>>> print string3
this is another string

>>>
```

Lists Python lists are similar to arrays in C or Java, except that the elements do not have to be of the same type. You can mix strings and integers, for example.

```
>>> a = ['spam', 'eggs', 100, 1234]
>>> a[0]
'spam'
>>>a[0] = 'ham'
>>>a
a = ['ham', 'eggs', 100, 1234]
```

You can iterate over the elements in a string by using a for-loop (note: there is a colon at the end of the for-statement; anything inside the for-statement is indented, and there is no `end` keyword).

```
for element in list:
   print element
```

You can exit for-loops with the `break` command:

```
for element in list:
   if element == 42:
      print "Found it: ", element
      break
   else:
      print "Still haven't found what I'm looking for: ", element
```

We can either append elements to a list, or we can extend a list with another list (appending all of its elements). When the argument of append is a list itself, we create a nested list:

```
>>> list = [1, 2, 3]
>>> list2 = [4, 5]
>>> list.append(list2)
```

```
>>> list
[1, 2, 3, [4, 5]]
>>> list.extend(list2)
>>> list
[1, 2, 3, [4, 5], 4, 5]
```

Tuples Tuples allow us to group elements together. They are formed with parentheses.

```
>>> student = "John"
>>> gpa = 4.0
>>> studentGradeTuple = (student, gpa)
>>> student1, gpa1 = studentGradeTuple
>>> student1
'John'
```

2.5.2 Dictionaries (associative arrays)

A Python dictionary maps keys to values. It can be declared with curly brackets. You can loop through the items by using the `iteritems()` operation:

```
>>> emptyDict = {}
>>> gpaDict = {"Mary":3.9, "Sue":4.0}
>>> gpaDict["John"] = 4.0
>>> print gpaDict["John"]
4.0
>>> print gpaDict['John']
4.0
>>> for student, gpa in gpaDict.iteritems():
...     print student, "\t", str(gpa)
```

Note: when you try things out in the Python shell, you still need to enter a line break at the end of the first line of the for statement, and then the prompt will change to `...`, but you will still have to indent the next line. The `str()` operation returns a string representation of its argument, and can be very useful for printing purposes.

We can also sort a dictionary, either by its keys or by its values (in ascending or descending order):

```
>>> from operator import itemgetter
>>> gpaDictSortedByKey = sorted(gpaDict.items())
>>> gpaDictSortedByKeyReverse = sorted(gpaDict.items(), reverse=True)
>>> gpaDictSortedByValue = sorted(gpaDict.items(),key=itemgetter(1))
>>> gpaDictSortedByValueReverse = sorted(gpaDict.items(),key=itemgetter(1), reverse=True)
```

2.5.3 Sets

When applied to a list, the `set()` operation returns a set (unordered collection) of all elements without duplicates:

```
>>> myList = [1, 2, 3, 2, 1, 3]
>>> mySet = set(myList)
>>> mySet
set([1, 2, 3])
```

2.5.4 Classes

Python also has classes, similar to Java. These allow you to create your own data types. We have defined a class Token, which represents the ith word in the jth sentence.

```
class Token:
    def __init__(self, s, w):
        self.sentence = s
        self.word = w
```

We can now represent the 5rd word in the 3th sentence as follows:

```
>>> token = Token(3, 5)
>>> token.sentence
3
>>> token.word
5
```

2.6 Your tasks

1. Start with the file `hw0_notNested.py`, which represents a corpus as a list of words. The ith word in the corpus is the element `corpus[i]` in the corpus list.
   (a) Write a function `countWords(corpus)` that counts the words in the corpus, and apply it to our corpus.
   (b) Write a function `getVocab(corpus)` that returns an alphabetically sorted list of the vocabulary used in the corpus, and apply it to our corpus.
   (c) We provide a function `createCorpusIndex(corpus)`, which maps each word to a list of all of its positions in the corpus at which it occurs. Write a function `printWordFrequencies(index, vocab)` that takes this index and the corpus vocabulary as arguments and prints out all words sorted by their frequencies (how often they occur in the corpus), in descending order (most frequent word first).
   (d) We provide a function `printConcordance(corpus, word_i)`, which prints out the word at position `word_i` in its context. Write a function `printCorpusConcordance(word, corpus, index)` that takes a word, the corpus and its index as arguments, and calls `printConcordance(corpus, word_i)` to print out all occurrences of `word` in the corpus.
2. Now use the file `hw0_Nested.py`, which represents each sentence as a list of words, and the corpus as a (nested) list of sentences. Here, the jth word in the ith sentences is the element`corpus[i][j]` of the nested corpus list.
   (a) Write a function `printStats(corpus)` that counts the sentences and words in the corpus, and apply it to our corpus.
   (b) Write a function `getVocab(corpus)` that returns an alphabetically sorted list of the vocabulary used in the corpus, and apply it to our corpus.
   (c) Write a function `createCorpusIndex_TupleVersion(corpus)`, which maps each word to a list of all of its positions in the corpus at which it occurs, representing the jth word in the ith sentences as a tuple.
   (d) Write a function `printWordFrequencies_TupleVersion(index, vocab)` that takes this tuple-based index and the corpus vocabulary as arguments and prints out all words sorted by their frequencies (how often they occur in the corpus), in descending order (most frequent word first).
   (e) We provide a function `printConcordance(corpus, word_i)`, which prints out the word at position `word_i` in its context. Write a function `printCorpusConcordance_TupleVersion(word, corpus, index)` that takes a word, the corpus and its tuple-based index as arguments, and calls `printConcordance(corpus, word_i)` to print out all occurrences of `word` in the corpus.
   (f) Write a function `createCorpusIndex_ClassVersion(corpus)`, which maps each word to a list of all of its positions in the corpus at which it occurs, representing the jth word in the ith sentences as an instance of class `Token`.
   (g) Write a function `printWordFrequencies_ClassVersion(index, vocab)` that takes this class-based index and the corpus vocabulary as arguments and prints out all words sorted by their frequencies (how often they occur in the corpus), in descending order (most frequent word first).
   (h) We provide a function `printConcordance(corpus, word_i)`, which prints out the word at position `word_i` in its context. Write a function `printCorpusConcordance_ClassVersion(word, corpus, index)` that takes a word, the corpus and its class-based index as arguments, and calls `printConcordance(corpus, word_i)` to print out all occurrences of `word` in the corpus.

2.7 Submission

The assignment will count two points towards your final HW grade (the other assignments are worth ten points each). The intent is for you to familiarize yourself with Python and Compass; as such, we won't grade your code for correctness (just do your best), but we do want to make sure that you know how to submit future assignments in the correct format. For the other assignments in the course you may be required to submit documentation along with your code (either a PDF report or a text README file); however, this is not necessary for this assignment.

2.7.1 What you will be graded on

The grade for Homework 0 will be based on the following rubric:

1 point: Submit your work on Compass (any non-empty submission will be worth at least one point).

1 point: Format your submission as described below, i.e. as a single archive that unpacks into the correct file structure: a directory named ***abc123*_cs447_HW0/**, containing your modified versions of `hw0_notNested.py` and `hw0_Nested.py` (*abc123* should be replaced by your NetID).

See below for further submission instructions.

2.7.2 What to submit

Submit your solution on Compass by navigating to "Course Content" →"Homeworks" →"Homework 0". The item in this folder will allow you to attach a compressed tarball with your solution; please include the following files:

`hw0_notNested.py`: your Python implementation of the first group of four tasks from Section 2.6.

`hw0_Nested.py`: your Python implementation of the second group of eight tasks from Section 2.6.

You may include the handout PDF and the `movies.txt` files if you like, but they aren't required.
To create the tarball, **save your files in a directory called *abc123*_cs447_HW0** (where *abc123* is your NetID) and then **create the archive from its parent directory** (`tar -czvf` *abc123*`_cs447_HW0.tar.gz` *abc123*`_cs447_HW0`).

Appendix: Installing Python

Running Python on the EWS Machines

To get started with the assignment using the Canopy distribution provided on the EWS machines, please read the following instructions.

Copying the files

First, send the `cs447_HW0.zip` archive to your home directory on the EWS machines, like so (replacing `ramusa2` with your NetID):

```
[ryan@hal9000 ~]$ scp cs447_HW0.zip ramusa2@remlnx.ews.illinois.edu:~
```

Then SSH into your account and unzip the archive:

```
[ryan@hal9000 ~]$ ssh ramusa2@remlnx.ews.illinois.edu
...
[ramusa2@linux-a1 ~]$ unzip cs447_HW0.zip
Archive:  cs447_HW0.zip
  inflating: cs447_HW0/hw0_Nested.py
  inflating: cs447_HW0/movies.txt
  inflating: cs447_HW0/hw0.pdf
  inflating: cs447_HW0/hw0_notNested.py
[ramusa2@linux-a1 ~]$ cd cs447_HW0
```

Next, load Canopy from the module system:

```
[ramusa2@linux-a1 cs447_HW0]$ module load canopy
[ramusa2@linux-a1 cs447_HW0]$ python
Enthought Canopy Python 2.7.6 | 64-bit | (default, Jun  4 2014, 16:32:15)
[GCC 4.1.2 20080704 (Red Hat 4.1.2-52)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Opening the Python interpreter let you verify that you're using the right version (Enthought Canopy Python 2.7.6). Close the shell (exit with Control-D or type `exit()`), and try running one of the stub modules we gave you:

```
[ramusa2@linux-a1 cs447_HW0]$ python hw0_notNested.py
reading file  movies.txt
Reading sentence 1000
Reading sentence 2000
Reading sentence 3000
Reading sentence 4000
Reading sentence 5000
Reading sentence 6000
Your task 1: count the total number of words (tokens) in our corpus
...
```

You're now ready to edit the stub files to implement the assignment. If you want to have the Canopy module pre-loaded for you every time you log in, execute the following command:

```
[ramusa2@linux-a1 cs447_HW0]$ echo "module load canopy" >> ~:.bashrc
```

Installing Python Yourself

For those of you that are interested in installing the python packages on your own machines rather than using the EWS machines, please see the following.

Canopy

The most straightforward method for installing the required packages for this course is through Canopy, which can be found here:

https://www.enthought.com/products/canopy/.

Selecting "Get Canopy" →"For Academics" →"Request Your License" will take you to a form where you can enter your `illinois.edu` email address and receive Canopy free for one year. Once provided with a license, please follow the provided installation instructions.

Separate package installation

The required packages can also be installed separately, if you prefer. Before anything else, Python must be installed. We recommend Python 2.7.6, which can be found here:

https://www.python.org/download/releases/2.7.6/

Please select your operating system and follow their installation instructions.
Once Python has been installed, the easiest way to install the required packages is with Pip, which can be found here:

https://pip.pypa.io/en/latest/installing.html

Once you follow the download and installation instructions on that page, please use the following commands (assuming a Unix system):

```
$ sudo pip install nltk
$ sudo pip install numpy
$ sudo pip install sklearn
```

These will automatically install NLTK (v3.0.4), numpy (v1.8.0rc1), and SciKit-Learn (v0.15.2). These closely correspond with the versions present in Canopy (NLTK 3.0.0b1, numpy 1.8.1, and SciKit-Learn 0.15.1).