Homework 4: Clustering of Books

Fundamentals of Data Science, 2016/2017

Note: to complete this homework you should master the basics of Python and NumPy covered in previous lessons – especially list comprehension, function definition, control loops, and multidimensional arrays. If you are not comfortable with these subjects, review them now.

In the following, "ID" denotes your student's ID if you have one, else it denotes your last name (please, suppress accents if present).

Your goal is to submit a Python module named `libID.py` and a Python script named `ID.py` satisfying the requirements described below.

1 `libID.py`

This module should implement four functions: `wordcount()`, `bag()`, `jaccard()`, `single_linkage()`.

1.1 `wordcount(filename)`

Compute a word count of the content of `filename`. The output must be a Python dictionary associating, to each word that appears in `filename`, the number of its occurrences. Words should be lower-cased, i.e. "Hello" and "HELLO" should both be counted as "hello". Also, any of the following characters, as well as whitespaces, should be treated as word delimiters and therefore should not appear in any word in the output (beware: the backslash `\` escapes the character after it):

```
<>/?;:,."'[]{}|=+-_()*&^%$#@!‘~\\\t\n\r
```

Example. The following code:

```
>>> import libID as mylib
>>> f = open('foo.txt', 'w')
>>> f.write('Pizza! PIZZA! pizza!\nI *like* writing "pizza!".\n\t\tThis is why I write this.')
>>> f.close()
>>> b = mylib.wordcount('foo.txt')
>>> print b
```

should give this output (the order is irrelevant):

```
{'like': 1, 'i': 2, 'is': 1, 'writing': 1, 'write': 1, 'this': 2, 'why': 1, 'pizza': 4}
```

1.2 `bag(wc, threshold=1)`

Return a bag of words from the given word count dictionary. The output should be a Python set containing all and only the words that occurred at least `threshold` times in the text file, according to the word count `wc`.

Example. Recalling the above example, the following code:

```
>>> import libID as mylib
>>> b = mylib.bag(mylib.wordcount('foo.txt'), threshold=2)
>>> print b
```

should give this output (the order is irrelevant):

```
{'i', 'this', 'pizza'}
```

1.3 `jaccard(s1, s2)`

Return the Jaccard *similarity* between two sets `s1` and `s2`.

Example. Recalling the previous example, the following code

```
>>> b = mylib.bag(mylib.wordcount('foo.txt'))
>>> mylib.jaccard(b.keys(), b.keys())
>>> mylib.jaccard(b.keys(), [])
>>> mylib.jaccard(b.keys(), b.keys()[:3])
```

should output these three lines

```
1.00
0.00
0.38
```

1.4 `single_linkage(D, k=2)`

Return a single-linkage clustering. `D` is a bidimensional, symmetric `numpy.ndarray` object where `D[i,j] = D[j,i]` is the *distance* (not the similarity!) between elements `i` and `j`. The optional parameter `k` specifies how many clusters to produce. The output should be an array or list whose i-th entry is the cluster ID of the i-th element; IDs can be arbitrary as long as the clustering is correct.

Example. The following code

```
>>> D = 0.1 * np.ones([5,5])
>>> D[:3,3:] += 0.6
>>> D[3:,:3] += 0.6
>>> D[np.diag_indices(len(D))] = 0.0
```

creates the matrix

```
[[ 0.   0.1  0.1  0.7  0.7]
 [ 0.1  0.   0.1  0.7  0.7]
 [ 0.1  0.1  0.   0.7  0.7]
 [ 0.7  0.7  0.7  0.   0.1]
 [ 0.7  0.7  0.7  0.1  0. ]]
```

Then

```
>>> mylib.single_linkage(D, 2)
```

should output

```
array([0, 0, 0, 3, 3])
```

Hints: you can use `numpy.unravel_index()` to get (row,column) indices from a "flat" index like the one returned by `.min()`. You can avoid two elements from being clustered together by setting their mutual distance to a large value. You can select a submatrix at the "intersection" of two lists of rows and columns by using `numpy.ndarray.ix()`.

2 ID.py

`ID.py` should take the names of n text documents as command line arguments:

```
$ python ID.py doc_0.txt doc_1.txt ... doc_n-1.txt
```

(hint: `sys.argv`, list slicing). The script should then:

1. compute a bag of words for each document, ignoring words occurring less than 10 times.
2. use `libID.jaccard()` to compute the n-by-n matrix `J` whose elements `J[i,j]` and `J[j,i]` equal the Jaccard *distance* (not the similarity) between documents i and j
3. cluster the documents according to `J`, using `libID.single_linkage()` with `k = 3`

The script should output the following, in this order and with no extra commentaries:

1. the sizes (number of elements) of the n bags of words in descending order
2. the distance matrix `J` with 2 decimal digits (hint: `np.set_printoptions()`)
3. the average of `J`'s elements
4. a list of lists, each containing the indices of the elements in a different cluster

Example. Invoked on 5 files,

```
$ python ID.py a.txt b.txt c.txt d.txt e.txt
```

you should get the following (actual numbers of course depend on the input file content):

```
[1738, 841, 470, 462, 253]
[[ 0.    0.95  0.95  0.9   0.24]
 [ 0.95  0.    0.97  0.97  0.95]
 [ 0.95  0.97  0.    0.75  0.95]
 [ 0.9   0.97  0.75  0.    0.9 ]
 [ 0.24  0.95  0.95  0.9   0.  ]]
0.682271650382
[[0, 4], [1], [2, 3]]
```

3 Submitting

You should submit the two Python files on or before Thursday November 10, 2016, 23:59 (Rome time), via email to the usual address, using the subject "ID hw4".