COMP 330/543 Lab #4: NumPy Performance Evaluation

At a high level, the purpose of this lab is to test how much more efficient Numpy bulk, vectorized operations can be than standard, dictionary-oriented Python computations

Your task is to compare three different solutions to the *co-occurrence problem*, a very fundamental computation in text analytics: Given a corpus of *n* documents and a dictionary containing a subset of the words in the corpus, we would like to compute the number of documents in the corpus that contain each possible $(word_i, word_i)$ pair where both $word_i$ and $word_j$ appear in the dictionary.

For example, let's say our documents are:

```
doc 1: {word1, word2, word4, word5}
doc 2: {word1, word2, word5}
doc 3: {word2, word3, word5}
```

Then the result of the co-occurrence computation is:

```
(word1, word1): 2 co-occurs
```
(this means word1 occurs twice in the corpus)
```
(word1, word2): 2 co-occurs
(word1, word4): 1 co-occurs
(word1, word5): 2 co-occurs
(word2, word2): 3 co-occurs
(word2, word3): 1 co-occurs
(word2, word4): 1 co-occurs
(word2, word5): 3 co-occurs
(word3, word3): 1 co-occurs
(word3, word5): 1 co-occurs
(word4, word4): 1 co-occurs
(word4, word5): 1 co-occurs
(word5, word5): 3 co-occurs
```

Task 1 - Dictionary-based Operations

First, run the pure dictionary-based LDA implementation we have provided to build a synthetic document corpus. This will build the `wordsInCorpus` object, which is a Python dictionary.

The dictionary's key is a document identifier, each key's value will be another dictionary. Each key in these inner dictionaries is a word identifier, their corresponding values are the number of occurrences of such words in the document.

Given this setup, write and execute the following code:

```
start = time.time()

# coOccurrences should be a map where the key is a (wordOne, wordTwo)
# pair and the value for each key is the number of times those two
# words co-occurred in a document.
# As such, the value will be a number between 0 and 50
coOccurences = {}

# Now, create a nested loop that fills up coOccurrences
# PUT YOUR CODE HERE

end = time.time()
print(end - start)
```

You should loop through all the documents. For each document, loop through all pairs of words and increment the count for that pair in `coOccurences`.

In case you don't remember, the way to loop through all of the keys in a Python dictionary is something like:

```
for wordOne in wordsInCorpus[doc]:
     # code here
```

Task 2 - Using NumPy Arrays

Another way to compute the same result is to use NumPy arrays. Note that the outer product of a vector x with itself creates a matrix where the entry at row `i` and column `j` is `x[i] * x[j]`.

Let's say we have a vector where the ith entry is 1 if the ith word in the dictionary appears in the document, and 0 otherwise. Then, if we take an outer product of this vector with itself, we will have a matrix where the entry at row *i*, column *j* is 1 if the document has both word *i* and word *j*.

Given this, write a code using NumPy that loops through all documents. Treat each document as a vector whose length is the length of the corpus. Then, you can use the outer product of the vector the word counts for that document with itself to create a matrix of co-occurrences for that document. Summing the 50 matrices gives us the answer.

Hint: In order to take a NumPy array foo and "clip" all of its entries so that none of them is greater than one, use the `clip` function, e.g., `np.clip(foo, 0, 1)`. Use the outer function to compute the outer product of two vectors, e.g., `np.outer(foo, bar)`.

To complete this task, first run the NumPy array-based LDA implementation that we have provided. This will build the `wordsInCorpus` object as a NumPy array. It stores the same data

as its dictionary counterpart. For example, `wordsInCorpus[34, 355]` is the number of times that the 355th word in the dictionary occurred in the 34th document.

As before, use the following code skeleton:

```
start = time.time()

# coOccurrences[i, j] will give the count of the number of times that
# word i and word j appear in the same document in the corpus
coOccurrences = np.zeros((2000,2000))

# Now, create a nested loop that fills up coOccurrences
# PUT YOUR CODE HERE

end = time.time()
print(end - start)
```

Task 3 - Using Matrix Multiplication

Finally, we can achieve the same result using matrix multiplications. In this case, write code using NumPy that uses a single matrix multiply to create coOccurrences. Note that you can use the `np.transpose` function to transpose a matrix and the `np.dot` function to multiply two matrices.

```
start = time.time()

# Now, create coOccurrences via Matrix multiplication
# PUT YOUR CODE HERE

end = time.time()
print(end - start)
```

What to Turn In

Submit the code for your three implementations to Canvas as **a single txt** file alongside the output you got from running each one.