Lab Number Three: Subarray Operators

The code below uses NumPy to implement the LDA model. “LDA” stands for “Latent Dirichlet Allocation”. LDA is popular a “topic model” often used in machine learning to model text documents. In the LDA model, each word in a document is produced by a so-called “topic”. Conceptually, a topic is some coherent idea that is found in multiple documents in a corpus. A topic could be “dangerous fishes of the world”, or “nice places to go on vacation”, or “famous battles of World War II”. Basically any coherent idea can be a topic.

```
import numpy as np

# there are 2000 words in the corpus
alpha = np.full (2000, .1)

# there are 100 topics
beta = np.full (100, .1)

# this gets us the probabilty of each word happening in each of the 100
topics
wordsInTopic = np.random.dirichlet (alpha, 100)

# produced [doc, topic, word] gives us the number of times that the given
word was
# produced by the given topic in the given doc
produced = np.zeros ((50, 100, 2000))

# generate each doc
for doc in range (0, 50):
        #
        # get the topic probabilities for this doc
        topicsInDoc = np.random.dirichlet (beta)
        #
        # assign each of the 2000 words in this doc to a topic
        wordsToTopic = np.random.multinomial (2000, topicsInDoc)
        #
        # and generate each of the 2000 words
        for topic in range (0, 100):
                produced[doc, topic] = np.random.multinomial
(wordsToTopic[topic], wordsInTopic[topic])
```

You don’t really need to understand what’s happening in this code. That’s not the point. The important thing is that this code produces the NumPy array called `produced`. `produced` stores the words present in a simulated document corpus, constructed by the model, consisting of 50 documents. `produced[doc, topic, word]` gives the number of times that the given word was produced by the given topic in the given doc. So if `produced[23, 19, 46]` is 7, it means that in document 23 in the corpus, the 19th topic produced 7 instances of word 46. Note that when the output is represented as an array like this, we typically assume that each word in the dictionary is mapped to a value from 0 to (the number of words in the dictionary minus one). Note that word 46 is the identifier for some word, but we don’t really care which one in this lab. In this

code, the corpus has 50 documents created by 100 different topics, and there are 2000 words in the dictionary.

All of that being said, the point of this lab is not to learn about statistical models for text. The point of the lab is to use NumPy to manipulate data stored in a NumPy array. In this lab, you need to complete the five tasks where we have not given an answer, and then post your answers to Canvas in order to get checked off. Start by running the above code, and then write the following, one-line codes (note that since the LDA code is running a program with pseudo-randomness, you can expect to get different results each time you run the code):

1. Write a line of code that computes the number of words produced by topic 17 in document 18. (one answer: `produced[18,17,:].sum ()`)
2. Write a line of code that computes the number of words produced by topic 17 thru 45 in document 18.
3. Write a line of code that computes the number of words in the entire corpus.
4. Write a line of code that computes the number of words in the entire corpus produced by topic 17 .
5. Write a line of code that computes the number of words in the entire corpus produced by topic 17 or topic 23. (one answer: `produced[:,np.array([17,23]),:].sum ()`)
6. Write a line of code that computes the number of words in the entire corpus produced by even numbered topics. (one answer: `produced[:,np.arange(0,100,2),:].sum ()`)
7. Write a line of code that computes the number of each word produced by topic 15.
8. Write a line of code that computes the topic responsible for the most instances of each word in the corpus.
9. Write a line of code that for each topic, computes the max number of occurrences (summed over all documents) of any word that it was responsible for. (nasty! One answer, though it is possible to compute up with a simpler answer: `produced[:,np.arange(0,100,1),produced.sum (0).argmax (1)].sum(0)`)