Programming Assignment 3 – Basic Probability

2020-2021, Master of Logic, University of Amsterdam

Teacher: Lina Murady
Coordinator: Wilker Aziz
Written by: Jakub Dotlačil

Submission deadline: Wednesday, November 18, 6 pm

Note: if the assignment is unclear to you or if you get stuck, do not hesitate to contact Lina. You can also post a question on Canvas.

1 Assignment

This week we will look at one of the most useful applications of natural language processing techniques: information retrieval. Suppose you have a vast *corpus* of documents such as books, articles or Wikipedia lemma's. You are looking for the document that most resembles your *query* "to be or not to be". How can we automatically retrieve the document that is 'closest to', or 'most similar to' or 'most relevant for' this query? That's what we look at today.

We will try to quantify the 'similarity' between each document and our query. The idea is to represent fragments of text (documents or queries) as *vectors*.[1] Vectors are simply lists of numbers, such as $(4, 4.2, 1.2, 6, 7)$. This particular vector was five-dimensional, since it has five entries. Vectors with two dimensions such as $(7, 2)$ can be thought of as points in a plane, or as an arrow pointing from the origin to that point in the plane. The 'similarity' between two vectors a and b can be measured by computing the *angle* between such vectors. Well, the cosine of that angle, to be precise, hence the name *cosine-similarity*. For now you only have to know that we can measure the similarity between two vectors (lists of numbers) $a = (a_1, a_2, \ldots, a_n)$ and $b = (b_1, b_2, \ldots, b_n)$ as follows:

$$\text{similarity}(a, b) = \frac{a_1 \cdot b_1 \ + \ a_2 \cdot b_2 \ + \ \ldots \ + \ a_n \cdot b_n}{\text{norm}(a) \cdot \text{norm}(b)} \tag{1}$$

where the norm (or length if you like) is

$$\text{norm}(a) = \sqrt{a_1^2 + a_2^2 + \cdots + a_n^2} \tag{2}$$

[1] You can read more about this so called vector-space model on Wikipedia

Good. So how do we turn text into vectors? There are many possible ways to do so, and we will only look at the simplest possibility (the most naive approach): use the vector of word-frequencies. This is not completely trivial. It is quite likely that a document contains a word that is not in any of the other documents, but to compare vectors they have to be of the same length. Differently put, they have to live in the same space. We therefore first construct a *vocabulary*: the list of the, say, 5000 most common words throughout the entire corpus. We can then count the number of times every word *in the vocabulary* occurs in a given document. If you put those 5000 frequencies in a list, you get a 5000-dimensional vector representing your document (or query). It's quite likely that it contains many zero's, but that's fine.[2] And once we have vectors, we can compute cosine similarities, and identify the document with the highest similarity to our query.

Word frequency vectors are probably the simplest, or most naive, vector representations. You can easily improve on this in many ways, and we encourage you to try so (but perhaps after finishing the homework). The Python file walks you through the assignment in nine steps, just like the in class exercises. The file contains lots of additional explanation, but if something is unclear, do not hesitate to post your questions on the forum. Here are some practical remarks:

- Note that you need to download and unzip the `corpus.zip` file and move the `corpus` folder to the folder you are working in.[3]
- You will have to use `Counter` objects a lot; see assignment 2 for more info.
- You are not allowed to use `numpy`, a library for working with vectors we will discuss later in the course. Part of this assignment is to learn how to work with Python's data structures. Also, it will illustrate why you normally *do* want to use `numpy` in these situations.
- Our implementation will not be very efficient (we read out all the files twice, for example), but we hope this structure makes it easier to follow *what* the code should be doing.

2 Grading

This assignment is graded stepwise: you can get a number of points for each of the nine steps. Below, we give some tips and tests that help you to check if the steps were solved correctly. If someone has made a mistake, you subtract points only once. For example suppose I've made a mistake in step 6, as a result of which step 7 also produces the wrong results. If step 7 would have worked fine if step 6 had been correct, then you can give full points for step 7. (But not for step 6).

[2] But note that if your vector contains *only* zeros, you might run into problems...
[3] On Canvas you can find how to download and unzip in Colab

1. **Collecting word frequencies: 1pt**

   Check if the corpus frequencies are computed correctly. The 'development subset' should contain 24164 words; the full corpus 146885.

2. **Make vocabulary: 1pt**

   Check if the vocabulary of the top 100 words is constructed correctly. The most common words in the corpus (full or subset) should be stop words like *the, and, of, to,* etc. Also the empty string `''` and the newline character `\n` should be highly frequent 'words'.

3. **Collect vocabulary word frequencies: 1pt**

   Check if the function `freq_to_vector` correctly returns a list of frequencies. You can test if `freq_to_vector(Counter({'foo': 10, 'bar': 2}), ['bar', 'foo']))` indeed returns `[2, 10]`. Note that the order of the frequencies has to correspond to the order of the words in the vocabulary.

4. **Collect vocabulary word frequencies: 1pt**

   You can check that, when using the development subset of the corpus, `corpus['1789-Washington']['freq_vect']` contains a vector of the form `[0, 116, 47, 71, ...]` indicating that *the* occurs 116 times, *and* 47 times and *of* 71 times.

5. **Norm: 1pt**

   Check if the function `norm` works as expected on the tests included in assignment file (e.g. `norm([3, 4])` returns `5.0`)

6. **Cosine similarity: 2pt**

   Check if the cosine similarity works as expected on the tests.

7. **Compute cosine similarities: 1pt**

   On the development subset, the similarity between two inaugural addresses should be 0.8873; between the addresses and Whitmans poetry collection 0.6428 and 0.6389 respectively.

8. **Arbitrary queries: 1pt**

   Check if the function correctly returns a frequency vector for arbitrary text. For example, `text_to_vector('foo foo bar test', ['bar', 'foo'])` should return `[1, 2]`.

9. **Rank documents: 1pt**

   On the development set, `1789-Washington` should be the most similar fragment, but on the full corpus we should correctly retrieve `1797-Adams`.