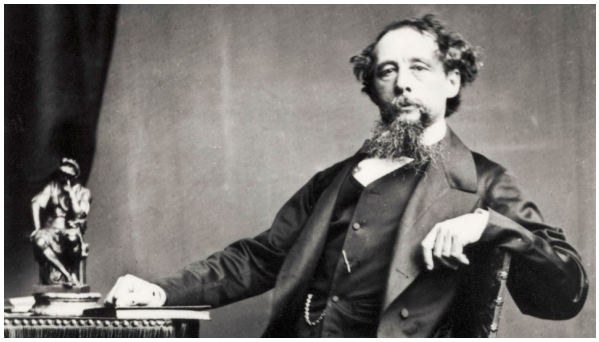
Lab 9: Authorship Detection



You're a historian who uncovered old transcripts from an unknown author. You want to align these texts with known authors and see if you can predict how likely they are to have come from them. This is actually a real task that real social scientists investigate. Some of Shakespeare's works are questioned as coming from him or not. In American Revolutionary times, people frequently published under pseudonyms to hide their identities, and we can use automated techniques to show similarities between known authors and their unattributed pieces of work. You can imagine modern day usefulness too, such as aligning anonymous social media posts with real people.

In this 2-part lab, you will process snippets from real novels with real authors, and then you will see if you can write a program to predict who wrote other unattributed snippets. For the purposes of this lab, we know who wrote all the snippets, and you will simply test against the correct answers.

Step 0: Directory Setup

1. Create a new folder for this lab (lab9)
2. Download the training novels, test novels, and short example snippets
   Or just copy/paste this command into your Ubuntu shell:

```
curl -O https://www.usna.edu/Users/cs/nchamber/courses/forall/lab/l09/train-snippets.tsv -O https://www.usna.e
```

3. Install the NLTK package to help with text processing:

```
sudo apt install python3-nltk
```

Week 1: Rudimentary Text Alignment

Our approach is to count words in each piece of text, and then compare the vector of word counts against an author's known texts. We'll find the best matching vector of word counts, and declare that to be the original author.

Shakespeare: "I do hate proud men as I do hate the engend'ring of toads."
Dickens: "The candle was burning low in the socket as he rose to his feet."
Unkown: "He was, altogether, as roystering and swaggering a young gentleman as ever stood four feet six."

Who wrote the last one? If you look at just word overlap ... we see more blue words matching Dickens ... which in this case is correct! In today's lab, you will write a program to do this word comparison automatically.

The key to your approach will be counting words! You can view each piece of text as a vector of word counts. See the counts below:

	I	do	hate	proud	men	as	the	engend'ring	of	toads	candle	was	burning	he	...	four	feet	six
Shakespeare	2	2	2	1	1	1	1	1	1	1	0	0	0	0	...	0	0	0
Dickens	0	0	0	0	0	1	1	0	0	0	1	1	1	1	...	0	1	0

Unknown	0	0	0	0	0	2	0	0	0	0	0	1	0	1	...	1	1	1

You will hold these counts in a Dictionary, of course. Each piece of text will have its own Dictionary of word counts. The keys are the words, and the values are the counts of each word. Above in blue are the words that appeared in more than one text, showing you the overlap. We'll use a metric to compare vectors for similarity, thus matching authors with their text.

Step 1: Read and Count

Create a file **part1-read.py**

The first step is to read text and store their word counts in a Dictionary.

Write a program that reads a file of text snippets (and their authors), and converts each snippet to a Dictionary of word counts. We had a homework which counted characters. You will now count words instead of characters. Use the NLTK library to help you split sentences into words:

```
import nltk
tokens = nltk.word_tokenize("I just can't even, for real.")
print(tokens)   # Prints:  ['I', 'just', 'ca', "n't", 'even', ',', 'for', 'real', '.']
```

One thing we don't want to do is count common words like 'the' and 'a'. The NLTK library helps us out because it comes with a handy-dandy "stop words list" for English. You can access it like this:

```
from nltk.corpus import stopwords
nltk.download('stopwords')
nltk.download('punkt')

# a List of words which you should use to NOT count
stops = stopwords.words('english')
```

Requirements:

1. **Write a function** called count_words(text) that takes a string argument, and returns a dictionary of word counts.
2. **Lowercase and don't count stop words** while counting.
3. **Write an entire program** that reads every line from a snippet file, and prints the author+dictionary to the terminal for each row in the snippet file. You can hard-code 'short-snippets.tsv' as your file.

Your output when running your program on short-snippets.tsv should look like this but your words will be in a different order when printed (that's ok, but counts should match!).

WARNING: you might discover on your own that NLTK has an object called *FreqDist* which counts words for you. We have not yet learned how to properly use new class types like this, so you are not allowed to use it in this lab.

Step 2: User Program to Match Author to User

Copy your Step 1 code to **part2-user.py**

Let's build on Step 1 and write a program that matches user text to their most similar author! The user will type in a sentence, and you'll tell them which author they are most like.

In order to do this, we will use your Part 1! Now change it so that instead of printing each author+dictionary, you save your authors and word count dictionaries in two lists:

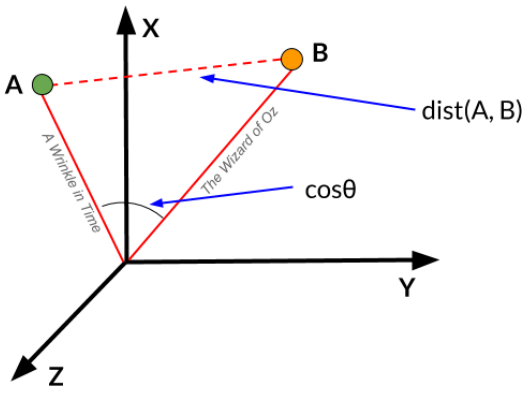
```
authors = [ 'DICKENS', 'DICKENS', 'AUSTEN', 'AUSTEN', 'HAWTHORNE', ... ]
counts = [ dict, dict, dict, dict, dict, ... ]
```

You'll read a sentence from the user, convert that to its own dictionary, and then loop over all your authors to see which dictionary is most similar. Sound easy?

In order to do this, we just need a mechanism to compare two Dictionaries, right? That's how we'll compare the user's text to an author's text. As we said above, you can think of these dictionaries as vectors where the cells are counts of words in English. Here is the Dickens text with the Unknown text again:

	I	do	hate	proud	men	as	the	engend'ring	of	toads	candle	was	burning	he	...	four	feet	six
Dickens	0	0	0	0	0	1	1	0	0	0	1	1	1	1	...	0	1	0
Unknown	0	0	0	0	0	2	0	0	0	0	0	1	0	1	...	1	1	1

You can see they both contain the words 'as', 'was', 'he', and 'feet'. Would you conclude that these two vectors are similar? *How* similar? How do we decide in a quantifiable way? Well we need a type of *similarity* metric that calculates the distance/similarity between two vectors. There are several options for this, such as Euclidean distance (dist in the image at right) or computing the cosine of the angle between them (cos in the image).



Cosine distance is commonly used because it normalizes the lengths of the vectors. The smaller the angle, the more word overlap between the texts. Cosine distance ranges from 1 (perfect word overlap) to 0 (no overlap). I am not requiring you to write this function, so instead I wrote it for you. Download the file at the link, and import it into your program!

```
import si286
d1 = dict()
d2 = dict()  # make sure to fill these, of course, with word counts
...
simscore = si286.cosine_sim(d1,d2)
```

If you have two dictionaries of word counts, you just call my cosine_sim() function as above, and it will give you the similarity score between two word vectors (two dictionaries of word counts).

Your task in this part is to read all texts from the file, save their Dictionary counts in a list, and then ask the user for a sentence. Find the best matching author to the user's sentence.

Requirements:

1. **Use train-snippets.tsv** now instead of short-snippets.tsv
2. **Save all lines** in the input file as Dictionaries. (make a List of Dictionaries)
3. **Ask the user for a sentence input**.
4. **Find the best match** against the user's input. Print it out with the author and the best passage that matched.

Required Output: (you don't need it to loop, can just run on one input and then stop)

```
Passage: The night is dark and full of terrors.
Most similar: SHAKESPEARE       Than death and honour. Let's to supper, come, And drown consideration.


Passage: It is not down on any map; true places never are.
Most similar: SHAW      CATHERINE (relenting). Ah! (Stretches her hand affectionately across the table to squeeze
```

```
Passage: Some hae meat and canna eat, -- And some wad eat that want it; But we hae meat, and we can eat, Sae let th
Most similar: CONRAD     chance--barring, of course, the killing him there and then, which wasn't so good, on accou
```

Step 3: Bulk Search Best Matching Authors

Copy part2-user.py to **part3-batch.py**

Your last program will now do full text matching, not from a user's small input. Instead you will read UNKNOWN texts from a file, and print out the best matching author. It's similar to Step 2, but you'll read a file of unknown texts instead of reading one sentence from user input.

Requirements:

1. **Load train-snippets.tsv as before as your main texts.**
2. **Read a test filename from the user**
3. **Read each line** from the test file, and find the best matching author text from train-snippets.tsv
4. **Print an author** for each line of the test file.

Required Output: (except for the '# wrong')

```
Test Filename: ten-snippets.tsv
File contains 10 lines.
HAWTHORNE           # wrong (Eliot)
JAMES
SHAKESPEARE
DICKENS
AUSTEN              # wrong (Eliot)
AUSTEN
TWAIN
CONRAD
AUSTEN
HAWTHORNE           # wrong (Hardy)
```

This rudimentary approach works fairly well! 7/10 correct if you matched our output. Next week we will explore some other approaches.

What to turn in

Visit the submit website and upload your THREE programs.