Course:	**INFO6143 Python**
Professor:	**Darryl Bedford**
Project:	**Lab #3**
Due Date:	**Friday, June 14, 2024 11:30 pm**
Submitting:	**Submit your zipped solution to the Lab 3 submission folder**

How will my lab be marked?

Marks Available	**What are the Marks Awarded For?**	**Mark Assigned**
1	Load the dictionary from the FastText100K.txt file	
1	Word entry loop works according to description below	
1	Keep track of the 5 highest cosine similarity values for each word entered	
1	Final display, duplicates the sample output shown in this PDF document	
1	Python file is **zipped** and submitted on-time to the submission folder	
5	Total	

Lab Description

Create a single Python program using the editor/IDE of your choice. Name this source file using the following template: your first name followed by an underscore and then "Lab3". For example, "Jim_Lab3.py".

For this lab, we'll only have a single requirement and as such we'll just create a Python app that can produce a duplicate of the sample output (see the console screen capture near the end of this description).

Overview

Now that you've had a chance to experiment with ways of determining meaning from natural languages such as English using a tool like WordNet, we want to try something that's more modern and is in some ways related to the inner workings of "Large Language Model" applications (LLMs) such as ChatGPT and GPT-4. More specifically, this refers to "word vectors" which is an approach that allows us to infer certain types of word relationships within large bodies of natural language text without having to do a lot of manual work to identify the meaning of those words.

You'll have some opportunities to work with these ideas in the NLP courses in both the second and third semesters of the AIM1 program but in the short term we'll do some experiments that will show you a few of the benefits of word vectors.

To begin with, we want to be able to analyze large bodies of text in a totally unsupervised way even if we can't infer the kind of precise dictionary meanings that we have in WordNet. In this case, we'll settle for being able to infer positive and negative correlations based on the proximity of words to other words in a piece of text.

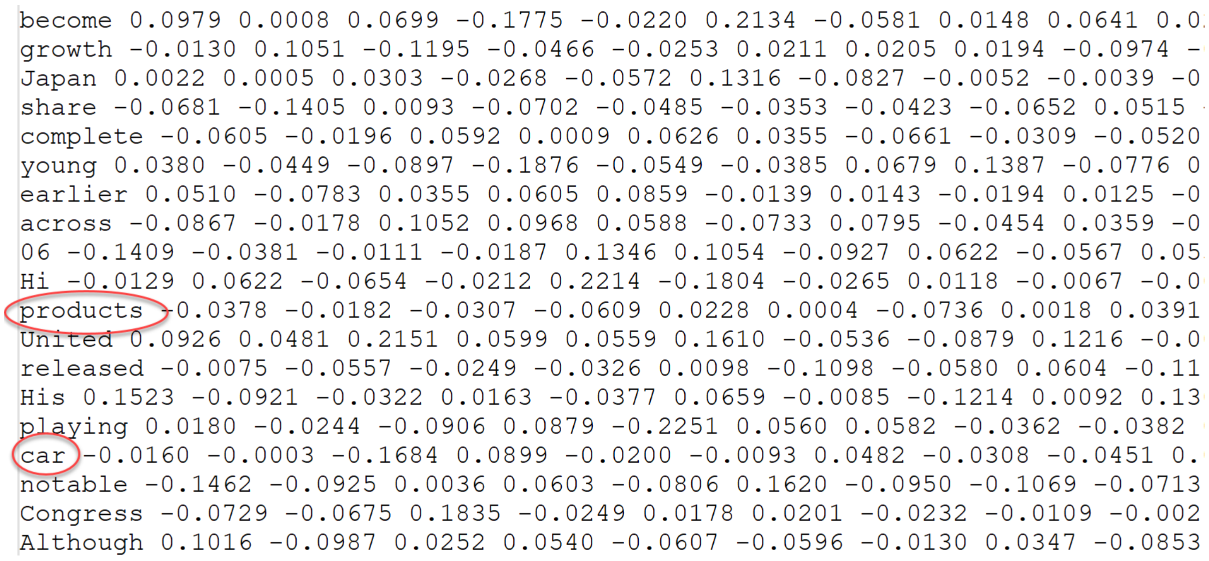
Companies such as Google, Facebook and Microsoft have built large "word vector" repositories that we can access to demonstrate some of these ideas.

Facebook uses a variation of word vectors called "fastText" which you can access using:

https://fasttext.cc/docs/en/english-vectors.html

We'll use a subset of their Wikipedia 2017 database (wiki-news-300d-1M.vec.zip) to try out some of these ideas.

Here's what a piece of the original word vector data looks like:

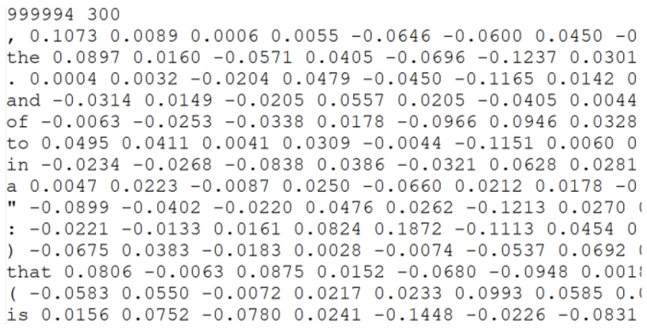


```
become 0.0979 0.0008 0.0699 -0.1775 -0.0220 0.2134 -0.0581 0.0148 0.0641 0.0
growth -0.0130 0.1051 -0.1195 -0.0466 -0.0253 0.0211 0.0205 0.0194 -0.0974 -
Japan 0.0022 0.0005 0.0303 -0.0268 -0.0572 0.1316 -0.0827 -0.0052 -0.0039 -0
share -0.0681 -0.1405 0.0093 -0.0702 -0.0485 -0.0353 -0.0423 -0.0652 0.0515
complete -0.0605 -0.0196 0.0592 0.0009 0.0626 0.0355 -0.0661 -0.0309 -0.0520
young 0.0380 -0.0449 -0.0897 -0.1876 -0.0549 -0.0385 0.0679 0.1387 -0.0776 0
earlier 0.0510 -0.0783 0.0355 0.0605 0.0859 -0.0139 0.0143 -0.0194 0.0125 -0
across -0.0867 -0.0178 0.1052 0.0968 0.0588 -0.0733 0.0795 -0.0454 0.0359 -0
06 -0.1409 -0.0381 -0.0111 -0.0187 0.1346 0.1054 -0.0927 0.0622 -0.0567 0.05
Hi -0.0129 0.0622 -0.0654 -0.0212 0.2214 -0.1804 -0.0265 0.0118 -0.0067 -0.0
products -0.0378 -0.0182 -0.0307 -0.0609 0.0228 0.0004 -0.0736 0.0018 0.0391
United 0.0926 0.0481 0.2151 0.0599 0.0559 0.1610 -0.0536 -0.0879 0.1216 -0.0
released -0.0075 -0.0557 -0.0249 -0.0326 0.0098 -0.1098 -0.0580 0.0604 -0.11
His 0.1523 -0.0921 -0.0322 0.0163 -0.0377 0.0659 -0.0085 -0.1214 0.0092 0.13
playing 0.0180 -0.0244 -0.0906 0.0879 -0.2251 0.0560 0.0582 -0.0362 -0.0382
car -0.0160 -0.0003 -0.1684 0.0899 -0.0200 -0.0093 0.0482 -0.0308 -0.0451 0.
notable -0.1462 -0.0925 0.0036 0.0603 -0.0806 0.1620 -0.0950 -0.1069 -0.0713
Congress -0.0729 -0.0675 0.1835 -0.0249 0.0178 0.0201 -0.0232 -0.0109 -0.002
Although 0.1016 -0.0987 0.0252 0.0540 -0.0607 -0.0596 -0.0130 0.0347 -0.0853
```

In the leftmost column is a single work or token (or sometimes a punctuation mark) and each token is followed by an array of floating-point numbers which together identify positive or negative proximity to other words within the original document.

The wiki-news-300d-1M.vec file contains a million words (tokens) and each vector contains 300 floating-point numbers. The only delimiter used is a blank space.

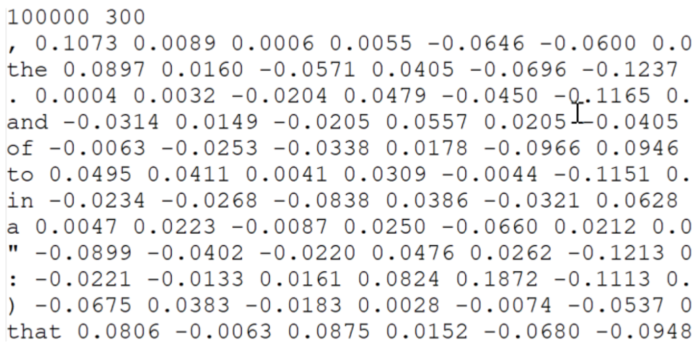
The beginning of the file looks like:



```
999994 300
, 0.1073 0.0089 0.0006 0.0055 -0.0646 -0.0600 0.0450 -0
the 0.0897 0.0160 -0.0571 0.0405 -0.0696 -0.1237 0.0301
. 0.0004 0.0032 -0.0204 0.0479 -0.0450 -0.1165 0.0142 0
and -0.0314 0.0149 -0.0205 0.0557 0.0205 -0.0405 0.0044
of -0.0063 -0.0253 -0.0338 0.0178 -0.0966 0.0946 0.0328
to 0.0495 0.0411 0.0041 0.0309 -0.0044 -0.1151 0.0060 0
in -0.0234 -0.0268 -0.0838 0.0386 -0.0321 0.0628 0.0281
a 0.0047 0.0223 -0.0087 0.0250 -0.0660 0.0212 0.0178 -0
" -0.0899 -0.0402 -0.0220 0.0476 0.0262 -0.1213 0.0270
: -0.0221 -0.0133 0.0161 0.0824 0.1872 -0.1113 0.0454 0
) -0.0675 0.0383 -0.0183 0.0028 -0.0074 -0.0537 0.0692
that 0.0806 -0.0063 0.0875 0.0152 -0.0680 -0.0948 0.001
( -0.0583 0.0550 -0.0072 0.0217 0.0233 0.0993 0.0585 0.
is 0.0156 0.0752 -0.0780 0.0241 -0.1448 -0.0226 -0.0831
```

As you can see, the first line in the file contains the number of lines in the repository followed by the row vector size. All tokens have the same number of floating-point values.

I’ve created a smaller version of this file which I’ve called “FastText100K.txt.” This version of the data has only 100,000 words, however it is still quite useful for testing because it loads faster.



```
100000 300
, 0.1073 0.0089 0.0006 0.0055 -0.0646 -0.0600 0.0
the 0.0897 0.0160 -0.0571 0.0405 -0.0696 -0.1237
. 0.0004 0.0032 -0.0204 0.0479 -0.0450 -0.1165 0.
and -0.0314 0.0149 -0.0205 0.0557 0.0205 -0.0405
of -0.0063 -0.0253 -0.0338 0.0178 -0.0966 0.0946
to 0.0495 0.0411 0.0041 0.0309 -0.0044 -0.1151 0.
in -0.0234 -0.0268 -0.0838 0.0386 -0.0321 0.0628
a 0.0047 0.0223 -0.0087 0.0250 -0.0660 0.0212 0.0
" -0.0899 -0.0402 -0.0220 0.0476 0.0262 -0.1213 0
: -0.0221 -0.0133 0.0161 0.0824 0.1872 -0.1113 0.
) -0.0675 0.0383 -0.0183 0.0028 -0.0074 -0.0537 0
that 0.0806 -0.0063 0.0875 0.0152 -0.0680 -0.0948
```

Detailed Steps for Doing Lab #3

The purpose of this lab is to create a Python application that accepts individual words from the “FastText100k.txt” file as keyboard entry and displays the 5 words from the file that “closely correlate” to the entered word, based on their cosine similarity calculation.

Begin by incorporating the code found in the “Coding Notes:” section of this lab description into your Python application.

Load the “FastText100K.txt” file into a Python dictionary and make sure you understand the resulting data (keys and values) in the dictionary.

Create a Python while loop that allows the user to enter individual words from the dictionary. For test purposes you can work with the words used in the sample output shown later in this PDF document, (enough, Obama, five, past). When the user enters a blank word, the application should end.

When the word has been entered, you’ll need to retrieve the 300-element vector for the entered word into a Python list.

You’ll then need to retrieve the vector for each word in the dictionary and as you do, invoke the cosine_similarity function to determine how close the word entered is to the word being tested.

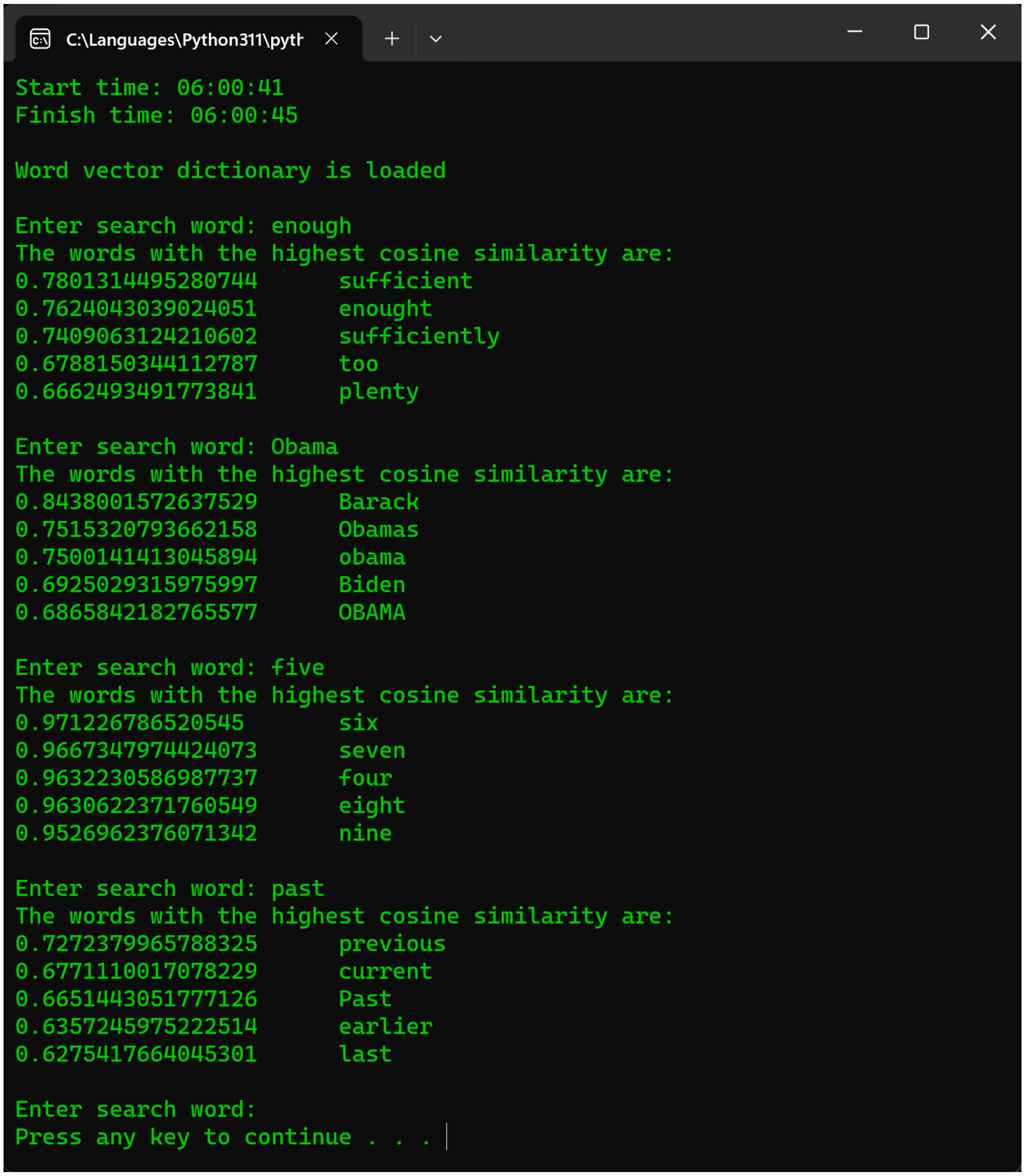
The cosine_similarity function will return a single double float value between 0.0 and 1.0. Larger numbers reflect a greater similarity between the two vectors. For the purpose of this lab, we won’t need to dive into how the math works.

As you look through all the words in the dictionary, keep track of the five largest cosine similarity numbers and their associated words as this will indicate the words that are “most similar” to your entered word.

When all the dictionary words have been tested, display the results as shown in the sample and prompt the user to enter the next word.

It's important when you're looping through the dictionary, to not test the word that was entered by the user as this will always have the highest score and we don't need to measure that.

Here's a sample output for this lab:



```
Start time: 06:00:41
Finish time: 06:00:45

Word vector dictionary is loaded

Enter search word: enough
The words with the highest cosine similarity are:
0.7801314495280744      sufficient
0.7624043039024051      enought
0.7409063124210602      sufficiently
0.6788150344112787      too
0.6662493491773841      plenty

Enter search word: Obama
The words with the highest cosine similarity are:
0.8438001572637529      Barack
0.7515320793662158      Obamas
0.7500141413045894      obama
0.6925029315975997      Biden
0.6865842182765577      OBAMA

Enter search word: five
The words with the highest cosine similarity are:
0.971226786520545       six
0.9667347974424073      seven
0.9632230586987737      four
0.9630622371760549      eight
0.9526962376071342      nine

Enter search word: past
The words with the highest cosine similarity are:
0.7272379965788325      previous
0.6771110017078229      current
0.6651443051777126      Past
0.6357245975222514      earlier
0.6275417664045301      last

Enter search word:
Press any key to continue . . .
```

Finally:

Submit your **zipped** Python source code to the Lab 3 submission folder.

Coding Notes:

```python
#####################################################
# This is a modified version of the function found at
# https://fasttext.cc/docs/en/english-vectors.html
# returns a dictionary which uses the word token
# as the key and a list of 300 floats as the value

import io
def load_vectors(fname):
    fin = io.open(fname, 'r', encoding='utf-8', newline='\n', errors='ignore')
    num_words, vec_size = map(int, fin.readline().split())
    data = {}
    for line in fin:
        tokens = line.rstrip().split(' ')
        data[tokens[0]] = list(map(float, tokens[1:]))

    return data

#######################################################
# These 3 functions calculate the cosine similarity for
# any 2 word vectors
def dot_product(vec_a, vec_b):
    dot_prod = 0.0;
    for i in range(len(vec_a)):
        dot_prod += vec_a[i] * vec_b[i]

    return dot_prod

import math
def magnitude(vector):
    return math.sqrt(dot_product(vector, vector))

# The entry point function
def cosine_similarity(vec_a, vec_b):
    dot_prod = dot_product(vec_a, vec_b)
    magnitude_a = magnitude(vec_a)
    magnitude_b = magnitude(vec_b)

    return dot_prod / (magnitude_a * magnitude_b)
```