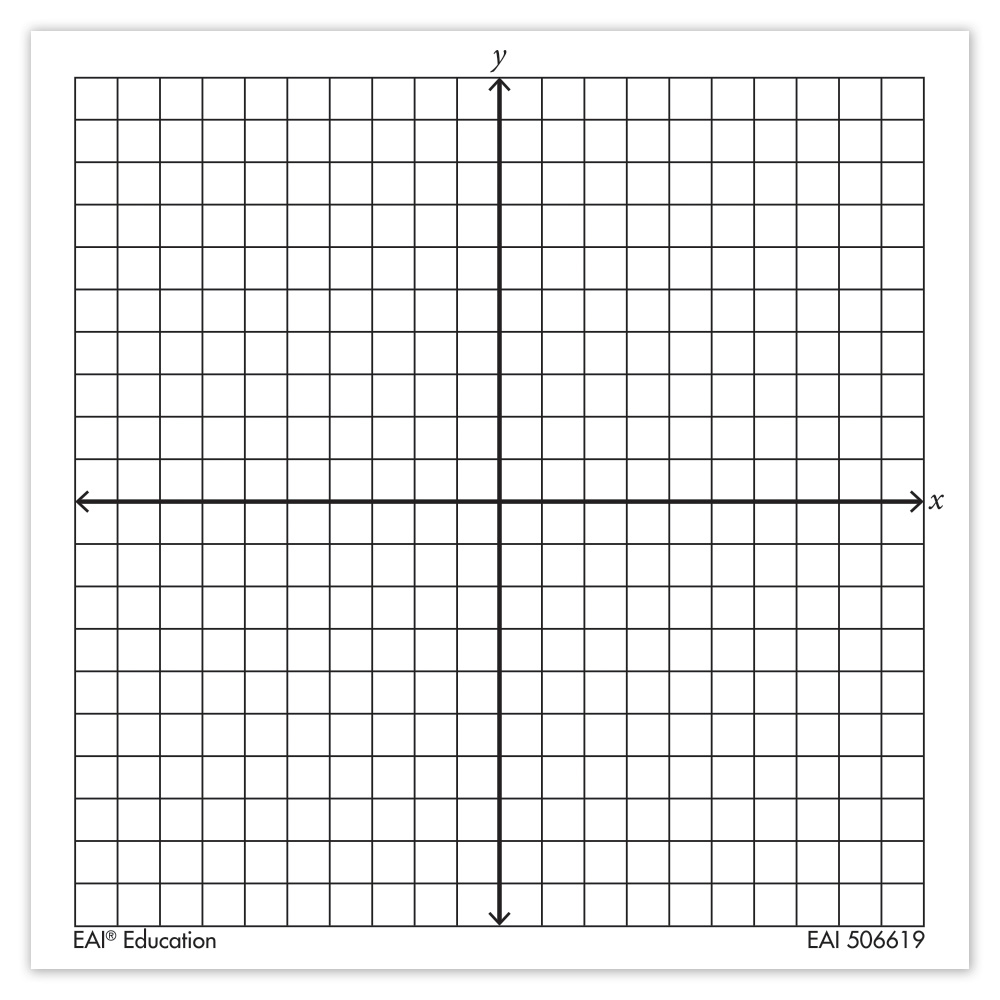
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | Nearest Cluster |
| (1, 2) | (1,2) | (−1, 5) | 0 | 5 |  |
| (2, 2) | (1,2) | (−1, 5) | 1 | 6 |  |
| (2, 1) | (1,2) | (−1, 5) | 2 | 7 |  |
| (−1, 5) | (1,2) | (−1, 5) | 5 | 0 |  |
| (−2, −1) | (1,2) | (−1, 5) | 6 | 7 |  |
| (−1, −1) | (1,2) | (−1, 5) | 4 | 6 |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | Nearest Cluster |
| (1, 2) |  | (−1, 5) | 2 | 5 |  |
| (2, 2) |  | (−1, 5) | 3 | 6 |  |
| (2, 1) |  | (−1, 5) | 2 | 7 |  |
| (−1, 5) |  | (−1, 5) | 6.8 | 0 |  |
| (−2, −1) |  | (−1, 5) | 5 | 7 |  |
| (−1, −1) |  | (−1, 5) | 4 | 6 |  |



x

x

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | Nearest Cluster |
| (1, 2) | (1,2) | (−2, -1) | 0 | 6 |  |
| (2, 2) | (1,2) | (−2, -1) | 1 | 7 |  |
| (2, 1) | (1,2) | (−2, -1) | 2 | 5 |  |
| (−1, 5) | (1,2) | (−2, -1) | 5 | 7 |  |
| (−2, −1) | (1,2) | (−2, -1) | 6 | 0 |  |
| (−1, −1) | (1,2) | (−2, -1) | 4 | 1 |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | Nearest Cluster |
| (1, 2) |  |  | 1/2 | 11/2 |  |
| (2, 2) |  |  | 3/2 | 13/2 |  |
| (2, 1) |  |  | 5/2 | 11/2 |  |
| (−1, 5) |  |  | 9/2 | 13/2 |  |
| (−2, −1) |  |  | 13/2 | 1/2 |  |
| (−1, −1) |  |  | 11/2 | 1/2 |  |

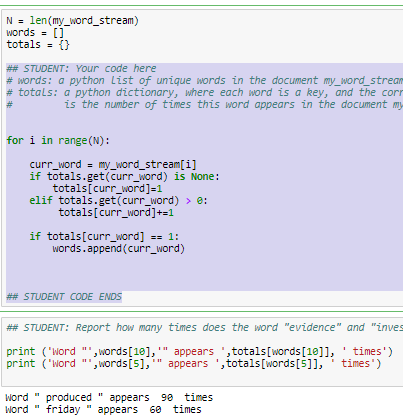
P1.

## STUDENT: Your code here

# words: a python list of unique words in the document my\_word\_stream as the vocabulary

# totals: a python dictionary, where each word is a key, and the corresponding value

# is the number of times this word appears in the document my\_word\_stream



for i in range(N):

curr\_word = my\_word\_stream[i]

if totals.get(curr\_word) is None:

totals[curr\_word]=1

elif totals.get(curr\_word) > 0:

totals[curr\_word]+=1

if totals[curr\_word] == 1:

words.append(curr\_word)

## STUDENT CODE ENDS

P2. ## STUDENT: Your code here

vocab\_words = [] # a list of words whose occurances (totals) are > 19

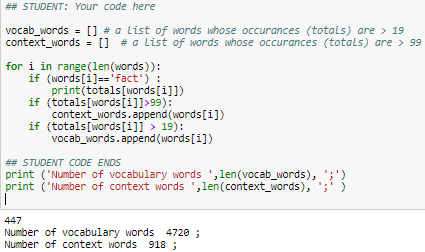
context\_words = [] # a list of words whose occurances (totals) are > 99

for i in range(len(words)):

if (words[i]=='fact') :

print(totals[words[i]])

if (totals[words[i]]>99):

 context\_words.append(words[i])

if (totals[words[i]] > 19):

vocab\_words.append(words[i])

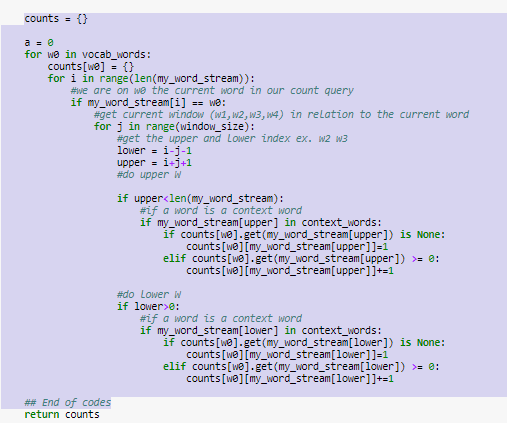
## STUDENT CODE ENDS

P3.

counts = {}

a = 0

for w0 in vocab\_words:

 counts[w0] = {}

for i in range(len(my\_word\_stream)):

#we are on w0 the current word in our count query

if my\_word\_stream[i] == w0:

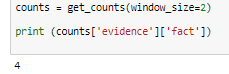
#get current window (w1,w2,w3,w4) in relation to the current word

for j in range(window\_size):

#get the upper and lower index ex. w2 w3

lower = i-j-1

upper = i+j+1

 #do upper W

if upper<len(my\_word\_stream):

#if a word is a context word

if my\_word\_stream[upper] in context\_words:

if counts[w0].get(my\_word\_stream[upper]) is None:

counts[w0][my\_word\_stream[upper]]=1

elif counts[w0].get(my\_word\_stream[upper]) >= 0:

counts[w0][my\_word\_stream[upper]]+=1

#do lower W

if lower>0:

#if a word is a context word

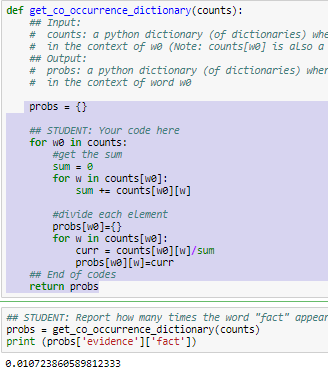
if my\_word\_stream[lower] in context\_words:

if counts[w0].get(my\_word\_stream[lower]) is None:

counts[w0][my\_word\_stream[lower]]=1

elif counts[w0].get(my\_word\_stream[lower]) >= 0:

counts[w0][my\_word\_stream[lower]]+=1

 ## End of codes

P4.

probs = {}

## STUDENT: Your code here

for w0 in counts:

#get the sum

sum = 0

for w in counts[w0]:

sum += counts[w0][w]

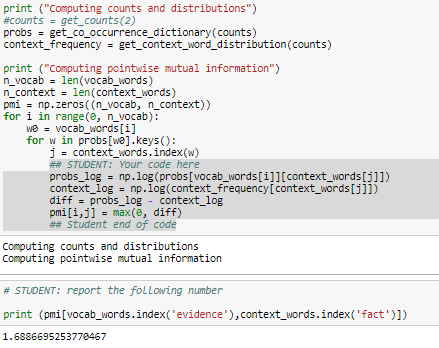
#divide each element

probs[w0]={}

for w in counts[w0]:

curr = counts[w0][w]/sum

probs[w0][w]=curr

 ## End of codes

return probs

P5.

## STUDENT: Your code here

probs\_log = np.log(probs[vocab\_words[i]][context\_words[j]])

context\_log = np.log(context\_frequency[context\_words[j]])

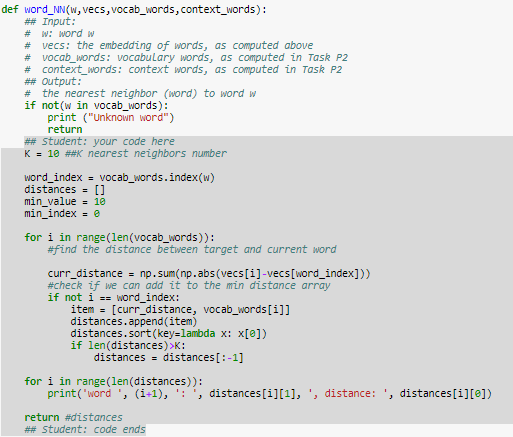
diff = probs\_log - context\_log

pmi[i,j] = max(0, diff)

## Student end of code

P6. ## Student: your code here

K = 10 ##K nearest neighbors number



word\_index = vocab\_words.index(w)

distances = []

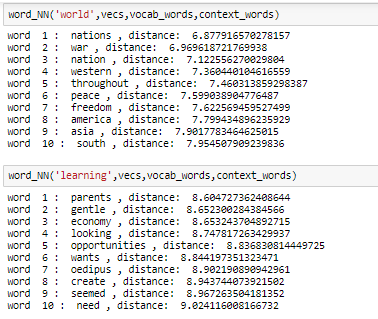
min\_value = 10

min\_index = 0

for i in range(len(vocab\_words)):

#find the distance between target and current word

curr\_distance = np.sum(np.abs(vecs[i]-vecs[word\_index]))

 #check if we can add it to the min distance array

if not i == word\_index:

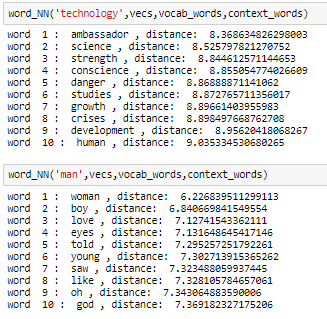
item = [curr\_distance, vocab\_words[i]]

distances.append(item)

distances.sort(key=lambda x: x[0])

if len(distances)>K:

distances = distances[:-1]



for i in range(len(distances)):

print('word ', (i+1), ': ', distances[i][1], ', distance: ',

distances[i][0])

return #distances

## Student: code ends

P7.

def find\_analogy(A,B,C,vecs,vocab\_words,context\_words):

## Input:

# A, B, C: words A, B, C

# vecs: the embedding of words, as computed above

# vocab\_words: vocabulary words, as computed in Task P2

# context\_words: context words, as computed in Task P2

## Output:

# the word that solves the analogy problem

## STUDENT: Your code here

## STUDENT: your code ends

K = 10 ##K nearest neighbors number

A\_index = vocab\_words.index(A)

B\_index = vocab\_words.index(B)

C\_index = vocab\_words.index(C)

#this i the target analogy we are trying to find

#i.e find the same distance

analogy\_vector = vecs[A\_index] - vecs[B\_index]

distances = []

analogys = []

min\_value = 10

min\_index = 0

## Student: your code here

print('word C :', C)

word\_NN(C,vecs,vocab\_words,context\_words)

#print

print('##find the closest analogy word')

for i in range(len(vocab\_words)):

#find the distance between target and current word

#check if we can add it to the min distance array

if not ((i == A\_index) or (i == B\_index) or (i == C\_index)):

temp\_distance = (vecs[i]-vecs[C\_index]) - (vecs[A\_index] - vecs[B\_index])

curr\_distance = np.sum(np.abs(temp\_distance))

item = [curr\_distance, vocab\_words[i]]

distances.append(item)

distances.sort(key=lambda x: x[0])

if len(distances)>K:

distances = distances[:-1]

#alalogy vector

print('Analogy Vector')

for i in range(len(distances)):

print('word ', (i+1), ': ', distances[i][1], ', distance: ', distances[i][0])

return

