Page No.	
Date	

IT-425 Natural Language Processing

Q1) a)

$$\begin{array}{c|ccccc} & \text{Python} & \text{Theory} & \text{Geometry} & \text{Worked} & \text{Research} \\ \text{doc1} & 1 & 0 & 0 & 0 & 0 \\ \text{doc2} & 0 & 1 & 1 & 1 & 1 \\ \text{doc3} & 0 & 1 & 0 & 1 & 1 \\ \text{doc4} & 0 & 0 & 0 & 1 & 0 \\ \text{doc5} & 1 & 0 & 0 & 0 & 0 \\ \text{doc6} & 0 & 0 & 0 & 0 & 1 \end{array}$$

We consider $\vec{\text{research}}$ as our text (Test) vector.

Now,

$$\text{cosine similarity}(u, v) = \frac{u \cdot v}{|\bar{u}||\bar{v}|}$$

$$\text{where } |\bar{u}| = \sqrt{\sum_{i=1}^{n} u_i^2}$$

$$\text{cosine}(\text{python}, \text{Research}) = \frac{(100010)^T (011001)}{|100010|\,|011001|}$$

$$= 0$$

Scanned with CamScanner

Page No.	
Date	

$$\text{cosine(theory, Research)} = \frac{(0,1,1,0,0,0)^T(0,1,1,0,0,1)^T}{|0,1,1,0,0,0|\,|0,1,1,0,0,1|}$$

$$\frac{1+1}{\sqrt{\sum u_i^2}\sqrt{\sum v_i^2}} = \frac{2}{|u||v|}$$

$$|u| = \sqrt{1+1} = 1.414$$

$$|v| = \sqrt{3} = 1.73$$

$$\text{CS(theory, research)} = \frac{2}{(1.41)(1.73)} = 0.8164$$

$$\text{CS(geometry, research)} = \frac{(0\,1\,0\,0\,0\,0)^T(0\,1\,1\,0\,0\,1)^T}{|0\,1\,0\,0\,0\,0|\,|0\,1\,1\,0\,0\,1|}$$

$$= \frac{1}{\sqrt{1}\sqrt{3}} = \frac{1}{1.73} = 0.5773$$

$$\text{cosine sim(worked, research)} = \frac{(0\,1\,1\,1\,0\,0)^T(0\,1\,1\,0\,0\,1)^T}{|0\,1\,1\,1\,0\,0|\,|0\,1\,1\,0\,0\,1|}$$

$$= \frac{2}{\sqrt{3}\sqrt{3}} = \frac{2}{3} = 0.667$$

Scanned with CamScanner

Page No.
Date

Now, we have found cosine similarity between the test vector research and all other vectors and the word closest to research is :-

Theory with cosine similarity
0.8164.

b) No, this is not a classification task as we have just represented our ~~we~~ words as vectors and that doesn't inherently classify ~~the~~ them.

Also, when we find similarity between 2 vectors, that doesn't automatically classify ~~the~~ them into categories.

To use these vector embeddings for classification, we need to define what our classes are and the factors which will decide where our vector will belong.

Let us say we wish to classify each word as positive ~~or~~ or negative then we can say that a vector whose norm is more than a specific value is positive or vica versa.

Scanned with CamScanner

We can even even use advanced classification techniques such as SVM's, Naive Baye's, Neural networks etc.

These methods will take our vector as an input and return the probability of being in one of the classes.

So we need to add a probabilistic training model or other training model to turn this into a classifier.

Q2) Rewriting the CV we obtain :-

	(data structures)	(coding Ninjas)	(Birkbeck College)
doc1	0	0	0
doc2	1	0	0
doc3	1	1	0
doc4	0	1	0
doc5	1	0	0
doc6	0	0	1

	(College, London)	(structures, Algorithms)
doc1	0	0
doc2	0	1
doc3	0	1
doc4	0	0
doc5	0	1
doc6	1	0

By choosing trigrams we obtain:-

	(London, plectrum, guitar)	(trinity, college, London)
doc1	0	0
doc2	0	0
doc3	0	0
doc4	0	0
doc5	0	0
doc6	1	1

	(data, structures, algorithms)	(plectrum, guitar, gu[illegible])
doc1	1	0
doc2	1	0
doc3	0	0
doc4	1	0
doc5	0	0
doc6	0	1

	(college, London, plectrum)
doc1	0
doc2	0
doc3	0
doc4	0
doc5	0
doc6	1

The resume is a small document with not many words and by using higher order n-grams like tri-grams we lose words that occur highly in unigram but not in trigram like Java or python or github.

So, we shouldn't use trigram and actually best result is delivered using unigram and bigram frequency.

We should prefer unigram over bigram as data like Java and Python is lost in bigrams, which is actually very important.

Hence, we should prefer unigram.

hot_vector.py × cosine_similarity.py × autocomplete.py × lemmatization.py × bigram.py × definit

```python
import nltk
import pandas
import pickle
from nltk.corpus import stopwords
from collections import Counter

stopwords_en = set(stopwords.words('english'))

# loading the document
resume = open('../assets/resume.txt', 'r')
document = resume.read().lower()
resume.close()

# tokenizing the document
tokenizer = nltk.RegexpTokenizer(r'\w+')
tokens = tokenizer.tokenize(document)

# removing stop words from tokens
tokens = [token for token in tokens if token not in stopwords_en and token.isalpha()]

# divide the document into n parts
documents = []
n = 6
k = len(tokens) // n
for i in range(n):
    documents.append(tokens[k * i: len(tokens) if i == n - 1 else (i + 1) * k])

def count_n_grams(data, n, start_token='<s>', end_token='<e>'):
    n_grams = {}
    for index, token in enumerate(data):
        n_gram = tuple(data[index: index + n])
        n_grams[n_gram] = n_grams.get(n_gram, 0) + 1
    return n_grams


bigrams = []
trigrams = []
for document in documents:
    bigrams.append(count_n_grams(tokens, 2))
    trigrams.append(count_n_grams(tokens, 3))


most_common_bi = set()
for bigram_freq in bigrams:
```

for token in most_common_bi › for index, bigram_freq in enume...

nsole Python Console 6: TODO

VCS Window Help nlp-playground - bigram.py - PyCharm

one_hot_vector.py | cosine_similarity.py | autocomplete.py | lemmatization.py | bigram.py | def

```python
def count_n_grams(data, n, start_token='<s>', end_token='<e>'):
    n_grams = {}
    for index, token in enumerate(data):
        n_gram = tuple(data[index: index + n])
        n_grams[n_gram] = n_grams.get(n_gram, 0) + 1
    return n_grams


bigrams = []
trigrams = []
for document in documents:
    bigrams.append(count_n_grams(tokens, 2))
    trigrams.append(count_n_grams(tokens, 3))


most_common_bi = set()
for bigram_freq in bigrams:
    for word, freq in Counter(bigram_freq).most_common(5):
        most_common_bi.add(word)


most_common_tri = set()
for bigram_freq in trigrams:
    for word, freq in Counter(bigram_freq).most_common(5):
        most_common_tri.add(word)

print(most_common_bi)
print(most_common_tri)


vectors_bi = {}
for token in most_common_bi:
    vector = [0] * 6
    for index, bigram_freq in enumerate(bigrams):
        vector[index] = int(token in bigram_freq)
    vectors_bi[token] = vector

print(vectors_bi)
```

for token in most_common_bi › for index, bigram_freq in enume...

Terminal | R Console | Python Console | 6: TODO

Page No.
Date

Q3) a) We can use wordnet to find all infections of a word that are norms and then all the different synonyms of a word and all these definitions eg. sample code:-

Code for finding Synonyms (Python)

```
from nltk.corpus import wordnet as wn.

word = 'dog'

synsets = wn. synsets(word, pos = 'n')

>> print(synsets[0].definition)

A member of genus Canis (.....)

for synset in synsets:
    print (synset.definition())
```

output

A member of Genus Canis...
A dull unattractive girl or woman
Informal Term for man
Someone who is Inormally....

(We get various synonyms)

Page No.
Date

(iii) We can even incorporate wordnet to identify lemmas and use it in lemmatization process.

b) Using wordnet we will find the closest synonyms and also consider closest synonyms. We will reduce all words into it's lowest common hypernyms and consider those words,

So for dog and cat we consider → Carnivore.

The ~~ratio~~ number of tokens will ~~increase~~ decrease but frequency will increase, it will lead to more generalization but loss of information.

Scanned with CamScanner