Hemanth Teja - HW1 - Question 4

February 6, 2020

1 TEXT RETRIEVAL SYSTEM USING Tf-idf AND COSINE SIMILARITY

1.1 NAME: HEMANTH TEJA (hy1713)

1.1.1 The goal of this problem is to implement a very simple text retrieval system. Given (as input) a database of documents as well as a query document (all provided in an attached.zip file), we implement a python program, to find the document in the database that is the best match to the query

The flow can be divided into 4 tasks :

1. Implementing a parser to read each document and convert it into a vector of words.

2. Compute tf-idf values for each word in every document as well as the query.

3. Compute the cosine similarity between tf-idf vectors of each document and the query.

4. Report the document with the maximum similarity value.

1.2 Theory Explanation

(Note : These explanations are pooled in from various online resources and textbooks)

Vector of Words: Vectors of numbers that represent the meaning of a word.

tf-idf : term frequency–inverse document frequency

Numerical statistic to reflect how important a word is to a document in a collection or corpus.

Cosine Similarity : Cosine similarity can be seen as a method of normalizing document length during comparison.

Language Used : Python

Python Modules Used - nltk, numpy, pandas, sklearn

```
In [1]: #NLTK-symbolic and statistical natural language processing.
        import nltk
        import os
        import string
        #Numpy- general-purpose array-processing package.
        import numpy as np
        import copy
        #Pandsa-data manipulation and analysis.
        import pandas as pd
        import pickle
        import re
        import math

        import sys
        !{sys.executable} -m pip install num2words
        from nltk.corpus import stopwords
        #A tokenizer that divides a string into substrings by splitting on the specified string
        from nltk.tokenize import word_tokenize
        #Stemmers remove morphological affixes from words, leaving only the word stem
        from nltk.stem import PorterStemmer
        #A Counter is a dict subclass for counting hashable objects
        from collections import Counter
        #num2words - Convert numbers to words in multiple languages
        from num2words import num2words
        #sklearn- machine learning library for the Python programming language
        #countvectorize-simple way to both tokenize a collection of text documents and
        ##build a vocabulary of known words
        from sklearn.feature_extraction.text import CountVectorizer
        from sklearn.metrics.pairwise import cosine_similarity
```

```
Collecting num2words
  Using cached https://files.pythonhosted.org/packages/eb/a2/ea800689730732e27711c41beed4b2a129b34974435b
Collecting docopt>=0.6.2 (from num2words)
Installing collected packages: docopt, num2words
Successfully installed docopt-0.6.2 num2words-0.5.10
```

1.3 Initially, we import the files from the zip file.

1.4 We navigate to the right directory and initialize the path.

```
In [2]: dir = '/home/hemanth/Desktop/Machine Learning/hw1q5'
        filenames = [dir+'/d1.txt', dir+'/d2.txt',dir+'/d3.txt',dir+'/d4.txt',dir+'/d5.txt']
```

1.5 We then initialize lists and dictonaries for processing the text data and storing them into vectors.

```
In [3]: files = {}

        ##We use this list to store every vector temporarily for output purposes.
        dict = []

        ##We use this list to store all the vector of words generated for each document
        Bag_of_Words = []
```

We open each file that needs to processed and validated. These files are from our local directory.

```
In [4]: for Curr_File in filenames:
            with open(Curr_File, "r") as file:
                if Curr_File in files:
                    continue
                files[Curr_File] = file.read()
```

1.6 Initializing functions for pre-processing

It is important to clean the text data of every file before we compute similarity.

Getting rid of unecessary words and punctuations can help overcome accuracy problems.

We implement 4 preprocessing modules to convert text to lower case, remove punctuations, remove apostrophe, and eliminate stop words.

```
In [5]: #Importing Stop Words Module of NLTK
        #Creating a list of banned stop words
        from nltk.corpus import stopwords
        import nltk
        nltk.download('stopwords')
        list_banned = stopwords.words('english')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /home/hemanth/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
In [6]: #Function to convert text to lower case
        def convert_lower_case(data):
            return np.char.lower(data)
```

```
In [7]: #Function to remove punctuations
        def remove_punctuation(data):
            symbols = "!\"#$%&()*+-./:;<=>?@[\]^_`{|}~\n"
            for i in range(len(symbols)):
                data = np.char.replace(data, symbols[i], ' ')
            data = np.char.replace(data, "  ", " ")
        data = np.char.replace(data, ',', '')
        return data
```

```
In [8]: #Function to remove apostrophe
        def  remove_apostrophe(data):
            return np.char.replace(data, "'", "")
```

```
In [9]: #Function that invokes the other preprocessing modules.
        def preprocess(text):
            text = remove_apostrophe(text)
            text = convert_lower_case(text)
            text = remove_punctuation(text)
            return text
```

1.7 Generating vector of words for each document

1.7.1 The output generated is in the form of a dictionary to represent the word and its frequency as pairs.

```
In [17]: for filename, text in files.items():
             with open (filename, "r") as myfile:
                 text=myfile.readlines()
                 #helps in attaining the file handle that is used in the output pointof view
                 head, tail = os.path.split(filename)

             #We process the files more for better precision
             for j in range(len(text)):
                 text[j] = re.sub(r'\W', ' ', text[j])
                 text[j] = re.sub(r'\s+', ' ', text[j])
             text= preprocess(text)
             print("=" * 100)
             print("VECTOR OF WORDS FOR :  " + tail)
             print("=" * 100)
             # create the transform
             vectorizer = CountVectorizer()
             # tokenize and build vocab
             vectorizer.fit(text)
             #Append the vocab to temporary store
             dict.append(vectorizer.vocabulary_)
             #Append each vector of a document to the list of all vectors
             Bag_of_Words.append(vectorizer.vocabulary_)
             # summarize
             print(dict)
             #Delete the temporary store
             dict.clear()
             print('\n')
```

```
========================================================================
VECTOR OF WORDS FOR :  d1.txt
```

```
==========================================================================
[{'java': 36, 'island': 34, 'indonesia': 31, 'with': 82, 'population': 60, 'million': 46, 'home': 29, 'percent': 58, 'indone


==========================================================================
VECTOR OF WORDS FOR :  d2.txt
==========================================================================
[{'java': 38, 'town': 85, 'approximately': 6, 'people': 59, 'georgia': 33, 'south': 79, 'ossetia': 56, 'according': 1, 'curre


==========================================================================
VECTOR OF WORDS FOR :  d3.txt
==========================================================================
[{'uss': 145, 'java': 73, 'wooden': 152, 'hulled': 67, 'sailing': 112, 'frigate': 55, 'united': 142, 'states': 125, 'navy': 83


==========================================================================
VECTOR OF WORDS FOR :  d4.txt
==========================================================================
[{'java': 63, 'coffee': 27, 'refers': 98, 'beans': 7, 'produced': 91, 'indonesian': 60, 'island': 61, 'phrase': 86, 'kopi': 65,


==========================================================================
VECTOR OF WORDS FOR :  d5.txt
==========================================================================
[{'java': 53, 'general': 43, 'purpose': 90, 'computer': 28, 'programming': 87, 'language': 55, 'concurrent': 29, 'class':
```

1.8 We now remove the STOP WORDS in each of the text files.

```
In [11]: dir = '/home/hemanth/Desktop/Machine Learning/hw1q5'
         filenames = [dir+'/d1.txt', dir+'/d2.txt',dir+'/d3.txt',dir+'/d4.txt',dir+'/d5.txt', dir+'/d_query.txt']
         append_docs= []

         for file in filenames:
             current_file = open(file,"r")
             current_document = current_file.read()
             import re
             current_document = re.sub(r'[^\w\s]','',current_document)
             current_document = re.sub('\s+', ' ', current_document).strip()


             temp_StoreDocx = current_document.split(" ")
             document_WithoutStopWords = [word for word in temp_StoreDocx if word not in list_banned]
             clean_document = ' '.join(document_WithoutStopWords)
```

```
    from sklearn.feature_extraction.text import TfidfVectorizer
    append_docs.append(clean_document)
```

The class DictVectorizer can be used to convert feature arrays represented as lists of standard Python dict objects to the NumPy/SciPy representation

```
In [12]: #Once we clean the document, we append to the list
         #We use the set of documents and inbuilt tf-idf functions to plot the values
         #This helps us compute the cosine similarity more efficiently
         final_text = np.array(append_docs)
         tfidf = TfidfVectorizer()
```

1.8.1 We build a vector of vectors that contain the tf-idf values of the documents

```
In [13]: build_tfidf_vectors = tfidf.fit_transform(final_text)
```

We now build a dataframe and use the "build_tfidf_vectors" that contains all the words and their tf-idf values for every document to generate the output

We can visualize the matrix as follows:

"build_tfidf_vectors" - Rows = tf-idf values of the word in a doc, Cols = The text document

```
In [14]: print("=" * 100)
         print("\n SAMPLE TF-IDF FOR d1.txt \n")
         print("=" * 100)
         print(pd.DataFrame(build_tfidf_vectors[0].toarray(), columns=tfidf.get_feature_names()))
         print("=" * 100)
         print("\n SAMPLE TF-IDF FOR d2.txt \n")
         print("=" * 100)
         print(pd.DataFrame(build_tfidf_vectors[1].toarray(), columns=tfidf.get_feature_names()))
         print("=" * 100)
         print("\n SAMPLE TF-IDF FOR d3.txt \n")
         print("=" * 100)
         print(pd.DataFrame(build_tfidf_vectors[2].toarray(), columns=tfidf.get_feature_names()))
         print("=" * 100)
         print("\n SAMPLE TF-IDF FOR d4.txt \n")
         print("=" * 100)
         print(pd.DataFrame(build_tfidf_vectors[3].toarray(), columns=tfidf.get_feature_names()))
         print("=" * 100)
         print("\n SAMPLE TF-IDF FOR d5.txt \n")
         print("=" * 100)
         print(pd.DataFrame(build_tfidf_vectors[4].toarray(), columns=tfidf.get_feature_names()))
```

```
=================================================================

 SAMPLE TF-IDF FOR d1.txt
```

```
=====================================================================
   1812  according  accused  acidity  acquired  active  administrative  after  \
0   0.0        0.0      0.0      0.0       0.0     0.0             0.0    0.0

   age  aged  ...  woodenhulled  wora     world  write  yards  years  yemen  \
0  0.0   0.0  ...           0.0   0.0  0.086515    0.0    0.0    0.0    0.0

   york  ytecode  zone
0   0.0      0.0   0.0

[1 rows x 477 columns]
=====================================================================

 SAMPLE TF-IDF FOR d2.txt

=====================================================================
   1812  according   accused  acidity  acquired  active  administrative  \
0   0.0   0.142036  0.071018      0.0       0.0     0.0        0.142036

      after  age  aged  ...  woodenhulled  wora  world  write  yards  \
0  0.058236  0.0   0.0  ...           0.0   0.0    0.0    0.0    0.0

   years  yemen  york  ytecode      zone
0    0.0    0.0   0.0      0.0  0.071018

[1 rows x 477 columns]
=====================================================================

 SAMPLE TF-IDF FOR d3.txt

=====================================================================
       1812  according  accused  acidity  acquired    active  administrative  \
0  0.066092        0.0      0.0      0.0       0.0  0.066092             0.0

      after  age  aged  ...  woodenhulled  wora  world  write     yards  \
0  0.054197  0.0   0.0  ...      0.066092   0.0    0.0    0.0  0.066092

   years  yemen      york  ytecode  zone
0    0.0    0.0  0.066092      0.0   0.0

[1 rows x 477 columns]
=====================================================================

 SAMPLE TF-IDF FOR d4.txt

=====================================================================
   1812  according  accused   acidity  acquired  active  administrative  \
0   0.0        0.0      0.0  0.065259       0.0     0.0             0.0
```

```
   after       age      aged  ...   woodenhulled  wora  world  write  yards  \
0    0.0  0.130517  0.130517  ...            0.0   0.0    0.0    0.0    0.0

      years     yemen  york  ytecode  zone
0  0.065259  0.065259   0.0      0.0   0.0

[1 rows x 477 columns]
================================================================================

 SAMPLE TF-IDF FOR d5.txt

================================================================================
   1812  according  accused  acidity  acquired  active  administrative  after  \
0   0.0        0.0      0.0      0.0  0.069419     0.0             0.0    0.0

   age  aged  ...   woodenhulled      wora  world     write  yards  years  \
0  0.0   0.0  ...            0.0  0.069419    0.0  0.069419    0.0    0.0

   yemen  york   ytecode  zone
0    0.0   0.0  0.069419   0.0

[1 rows x 477 columns]
```

```
In [15]: tfidf.get_feature_names()
         output = pd.DataFrame(build_tfidf_vectors.toarray(), columns=tfidf.get_feature_names())
```

1.8.2 SKLEARN - Helps Compute cosine similarity between samples in X and Y.

Cosine similarity, or the cosine kernel, computes similarity as the normalized dot product of X and Y.

```
In [16]: from sklearn.metrics.pairwise import cosine_similarity
         #build_tfidf_vectors[-1] = refers to the d_query vector of tf-idf values
         cosine = cosine_similarity(build_tfidf_vectors[-1],build_tfidf_vectors)
         print("The Similarity in Percenrtages Are:")
         print(cosine*100)
         print('\n')
         print("Note: The last value is 100- because its the similarity of d_query with itself")
```

```
The Similarity in Percenrtages Are:
[[  8.72767967   6.14085898   6.66743192  35.46772154  12.00517919
  100.        ]]


Note: The last value is 100- because its the similarity of d_query with itself
```

2 From the cosine output, we can conclude that - the fourth value is the highest in terms of percentage

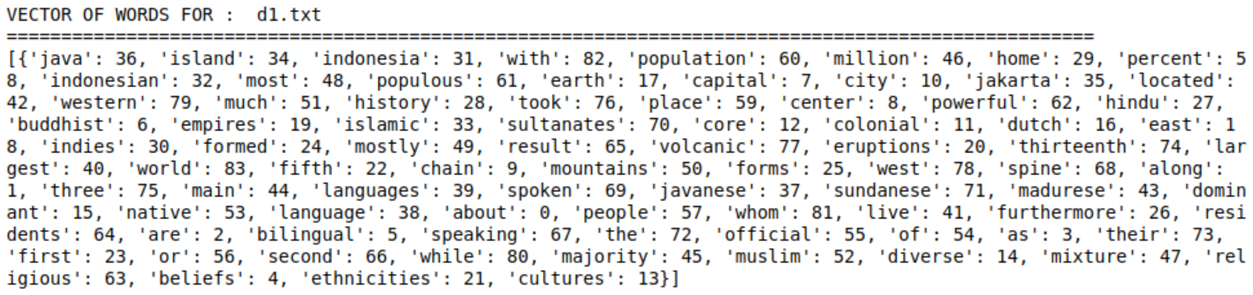
2.1 This value belongs to "d4.txt"

2.2 The percentage of similarity = 35.46 %

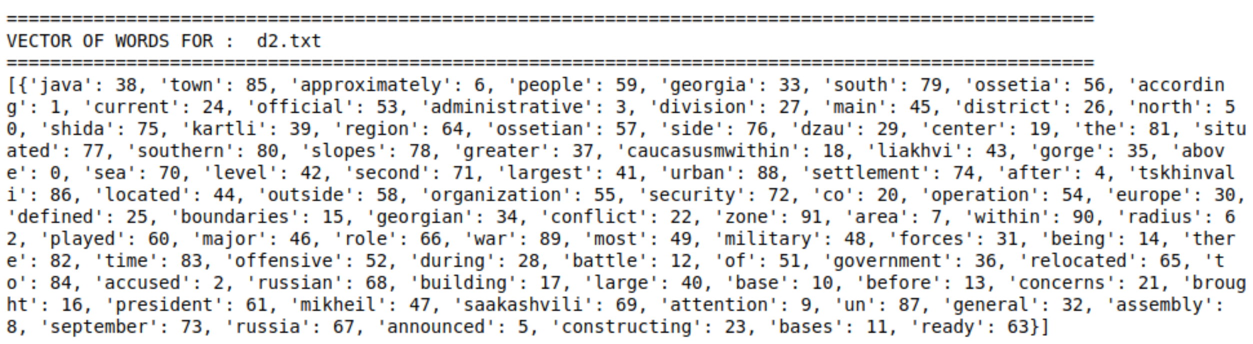
2.3 Therefore the document with maximum similarity is : "d4.txt "

In [ ]:

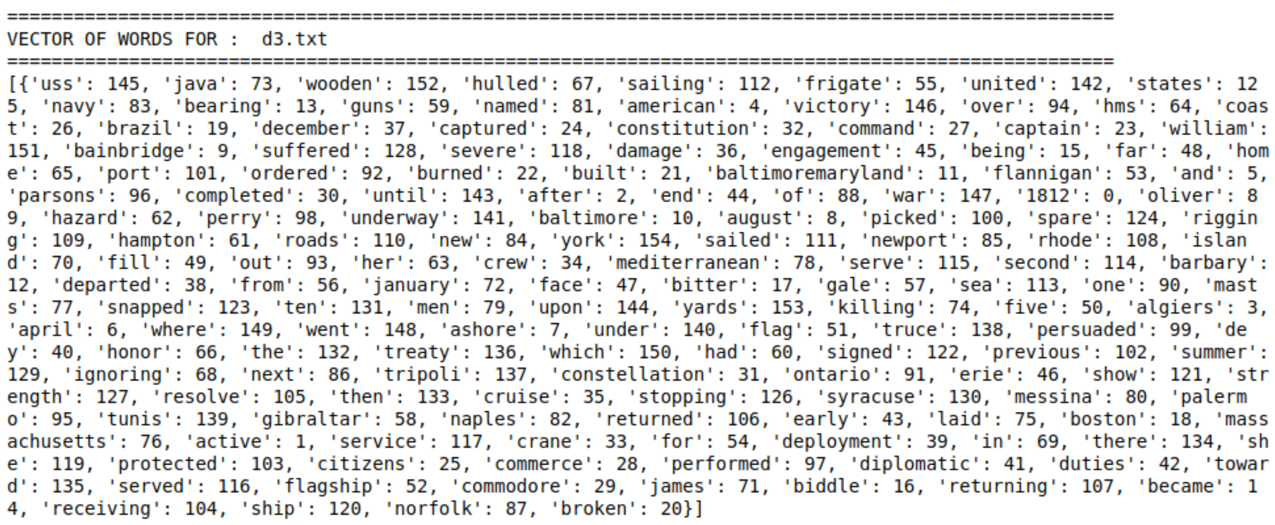
COMPLETE OUTPUT OF THE VECTOR OF WORDS – OUTPUT SCREENSHOTTED FROM JUPYTER NOTEBOOK



```
VECTOR OF WORDS FOR :  d1.txt
==================================================================================================
[{'java': 36, 'island': 34, 'indonesia': 31, 'with': 82, 'population': 60, 'million': 46, 'home': 29, 'percent': 5
8, 'indonesian': 32, 'most': 48, 'populous': 61, 'earth': 17, 'capital': 7, 'city': 10, 'jakarta': 35, 'located':
42, 'western': 79, 'much': 51, 'history': 28, 'took': 76, 'place': 59, 'center': 8, 'powerful': 62, 'hindu': 27,
'buddhist': 6, 'empires': 19, 'islamic': 33, 'sultanates': 70, 'core': 12, 'colonial': 11, 'dutch': 16, 'east': 1
8, 'indies': 30, 'formed': 24, 'mostly': 49, 'result': 65, 'volcanic': 77, 'eruptions': 20, 'thirteenth': 74, 'lar
gest': 40, 'world': 83, 'fifth': 22, 'chain': 9, 'mountains': 50, 'forms': 25, 'west': 78, 'spine': 68, 'along':
1, 'three': 75, 'main': 44, 'languages': 39, 'spoken': 69, 'javanese': 37, 'sundanese': 71, 'madurese': 43, 'domin
ant': 15, 'native': 53, 'language': 38, 'about': 0, 'people': 57, 'whom': 81, 'live': 41, 'furthermore': 26, 'resi
dents': 64, 'are': 2, 'bilingual': 5, 'speaking': 67, 'the': 72, 'official': 55, 'of': 54, 'as': 3, 'their': 73,
'first': 23, 'or': 56, 'second': 66, 'while': 80, 'majority': 45, 'muslim': 52, 'diverse': 14, 'mixture': 47, 'rel
igious': 63, 'beliefs': 4, 'ethnicities': 21, 'cultures': 13}]
```



```
==================================================================================================
VECTOR OF WORDS FOR :  d2.txt
==================================================================================================
[{'java': 38, 'town': 85, 'approximately': 6, 'people': 59, 'georgia': 33, 'south': 79, 'ossetia': 56, 'accordin
g': 1, 'current': 24, 'official': 53, 'administrative': 3, 'division': 27, 'main': 45, 'district': 26, 'north': 5
0, 'shida': 75, 'kartli': 39, 'region': 64, 'ossetian': 57, 'side': 76, 'dzau': 29, 'center': 19, 'the': 81, 'situ
ated': 77, 'southern': 80, 'slopes': 78, 'greater': 37, 'caucasusmwithin': 18, 'liakhvi': 43, 'gorge': 35, 'abov
e': 0, 'sea': 70, 'level': 42, 'second': 71, 'largest': 41, 'urban': 88, 'settlement': 74, 'after': 4, 'tskhinval
i': 86, 'located': 44, 'outside': 58, 'organization': 55, 'security': 72, 'co': 20, 'operation': 54, 'europe': 30,
'defined': 25, 'boundaries': 15, 'georgian': 34, 'conflict': 22, 'zone': 91, 'area': 7, 'within': 90, 'radius': 6
2, 'played': 60, 'major': 46, 'role': 66, 'war': 89, 'most': 49, 'military': 48, 'forces': 31, 'being': 14, 'ther
e': 82, 'time': 83, 'offensive': 52, 'during': 28, 'battle': 12, 'of': 51, 'government': 36, 'relocated': 65, 't
o': 84, 'accused': 2, 'russian': 68, 'building': 17, 'large': 40, 'base': 10, 'before': 13, 'concerns': 21, 'broug
ht': 16, 'president': 61, 'mikheil': 47, 'saakashvili': 69, 'attention': 9, 'un': 87, 'general': 32, 'assembly':
8, 'september': 73, 'russia': 67, 'announced': 5, 'constructing': 23, 'bases': 11, 'ready': 63}]
```



```
==================================================================================================
VECTOR OF WORDS FOR :  d3.txt
==================================================================================================
[{'uss': 145, 'java': 73, 'wooden': 152, 'hulled': 67, 'sailing': 112, 'frigate': 55, 'united': 142, 'states': 12
5, 'navy': 83, 'bearing': 13, 'guns': 59, 'named': 81, 'american': 4, 'victory': 146, 'over': 94, 'hms': 64, 'coas
t': 26, 'brazil': 19, 'december': 37, 'captured': 24, 'constitution': 32, 'command': 27, 'captain': 23, 'william':
151, 'bainbridge': 9, 'suffered': 128, 'severe': 118, 'damage': 36, 'engagement': 45, 'being': 15, 'far': 48, 'hom
e': 65, 'port': 101, 'ordered': 92, 'burned': 22, 'built': 21, 'baltimoremaryland': 11, 'flannigan': 53, 'and': 5,
'parsons': 96, 'completed': 30, 'until': 143, 'after': 2, 'end': 44, 'of': 88, 'war': 147, '1812': 0, 'oliver': 8
9, 'hazard': 62, 'perry': 98, 'underway': 141, 'baltimore': 10, 'august': 8, 'picked': 100, 'spare': 124, 'riggin
g': 109, 'hampton': 61, 'roads': 110, 'new': 84, 'york': 154, 'sailed': 111, 'newport': 85, 'rhode': 108, 'islan
d': 70, 'fill': 49, 'out': 93, 'her': 63, 'crew': 34, 'mediterranean': 78, 'serve': 115, 'second': 114, 'barbary':
12, 'departed': 38, 'from': 56, 'january': 72, 'face': 47, 'bitter': 17, 'gale': 57, 'sea': 113, 'one': 90, 'mast
s': 77, 'snapped': 123, 'ten': 131, 'men': 79, 'upon': 144, 'yards': 153, 'killing': 74, 'five': 50, 'algiers': 3,
'april': 6, 'where': 149, 'went': 148, 'ashore': 7, 'under': 140, 'flag': 51, 'truce': 138, 'persuaded': 99, 'de
y': 40, 'honor': 66, 'the': 132, 'treaty': 136, 'which': 150, 'had': 60, 'signed': 122, 'previous': 102, 'summer':
129, 'ignoring': 68, 'next': 86, 'tripoli': 137, 'constellation': 31, 'ontario': 91, 'erie': 46, 'show': 121, 'str
ength': 127, 'resolve': 105, 'then': 133, 'cruise': 35, 'stopping': 126, 'syracuse': 130, 'messina': 80, 'palerm
o': 95, 'tunis': 139, 'gibraltar': 58, 'naples': 82, 'returned': 106, 'early': 43, 'laid': 75, 'boston': 18, 'mass
achusetts': 76, 'active': 1, 'service': 117, 'crane': 33, 'for': 54, 'deployment': 39, 'in': 69, 'there': 134, 'sh
e': 119, 'protected': 103, 'citizens': 25, 'commerce': 28, 'performed': 97, 'diplomatic': 41, 'duties': 42, 'towar
d': 135, 'served': 116, 'flagship': 52, 'commodore': 29, 'james': 71, 'biddle': 16, 'returning': 107, 'became': 1
4, 'receiving': 104, 'ship': 120, 'norfolk': 87, 'broken': 20}]
```

```
=================================================================================================
VECTOR OF WORDS FOR :  d4.txt
=================================================================================================
[{'java': 63, 'coffee': 27, 'refers': 98, 'beans': 7, 'produced': 91, 'indonesian': 60, 'island': 61, 'phras
e': 86, 'kopi': 65, 'origin': 81, 'but': 18, 'used': 128, 'distinguish': 35, 'style': 112, 'strong': 111, 'bl
ack': 10, 'very': 130, 'sweet': 116, 'some': 106, 'countries': 31, 'can': 20, 'refer': 97, 'general': 49, 'pr
imarily': 88, 'grown': 53, 'large': 66, 'estates': 39, 'built': 16, 'dutch': 38, 'century': 22, 'five': 42,
'largest': 67, 'blawan': 12, 'also': 6, 'spelled': 108, 'belawan': 8, 'or': 80, 'blauan': 11, 'jampit': 62,
'djampit': 36, 'pancoer': 84, 'pancur': 85, 'kayumas': 64, 'tugosari': 126, 'they': 122, 'cover': 32, 'more':
73, 'hectares': 57, 'transport': 125, 'ripe': 102, 'cherries': 24, 'quickly': 95, 'their': 119, 'mills': 71,
'after': 1, 'harvest': 54, 'pulp': 93, 'then': 120, 'fermented': 40, 'washed': 131, 'off': 76, 'using': 129,
'wet': 132, 'process': 90, 'rigorous': 101, 'quality': 94, 'control': 30, 'results': 100, 'with': 135, 'goo
d': 50, 'heavy': 56, 'body': 14, 'overall': 82, 'impression': 59, 'sometimes': 107, 'rustic': 103, 'flavor':
43, 'profiles': 92, 'display': 34, 'lasting': 68, 'finish': 41, 'best': 9, 'smooth': 105, 'supple': 115, 'hav
e': 55, 'subtle': 113, 'herbaceous': 58, 'note': 75, 'taste': 118, 'prized': 89, 'one': 79, 'component': 29,
'traditional': 124, 'mocca': 72, 'blend': 13, 'which': 133, 'pairs': 83, 'from': 47, 'yemen': 137, 'certain':
23, 'age': 2, 'portion': 87, 'for': 46, 'years': 136, 'normally': 74, 'burlap': 17, 'sacks': 104, 'regularl
y': 99, 'aired': 4, 'dusted': 37, 'flipped': 45, 'turn': 127, 'green': 52, 'light': 69, 'brown': 15, 'gains':
48, 'strength': 110, 'while': 134, 'losing': 70, 'acidity': 0, 'aged': 3, 'coffees': 28, 'flavors': 44, 'rang
ing': 96, 'cedar': 21, 'spices': 109, 'such': 114, 'cinnamon': 25, 'clove': 26, 'often': 77, 'develop': 33,
'thick': 123, 'almost': 5, 'syrupy': 117, 'these': 121, 'called': 19, 'old': 78, 'government': 51}]


=================================================================================================
VECTOR OF WORDS FOR :  d5.txt
=================================================================================================
[{'java': 53, 'general': 43, 'purpose': 90, 'computer': 28, 'programming': 87, 'language': 55, 'concurrent':
29, 'class': 18, 'based': 12, 'object': 72, 'oriented': 76, 'specifically': 100, 'designed': 34, 'few': 39,
'implementation': 49, 'dependencies': 32, 'possible': 85, 'intended': 51, 'let': 57, 'application': 7, 'devel
opers': 36, 'write': 120, 'once': 73, 'run': 97, 'anywhere': 5, 'wora': 119, 'meaning': 66, 'compiled': 23,
'code': 21, 'can': 17, 'all': 1, 'platforms': 82, 'support': 105, 'without': 118, 'the': 109, 'need': 71, 're
compilation': 91, 'applications': 8, 'typically': 110, 'bytecode': 16, 'virtual': 113, 'machine': 63, 'jvm':
54, 'regardless': 93, 'architecture': 9, 'one': 74, 'most': 69, 'popular': 84, 'languages': 56, 'use': 112,
'particularly': 80, 'client': 20, 'server': 98, 'web': 115, 'with': 117, 'reported': 96, 'million': 68, 'wa
s': 114, 'originally': 78, 'developed': 35, 'by': 15, 'james': 52, 'gosling': 45, 'at': 11, 'sun': 104, 'micr
osystems': 67, 'which': 116, 'has': 46, 'since': 99, 'been': 13, 'acquired': 0, 'oracle': 75, 'corporation':
31, 'and': 4, 'released': 94, 'core': 30, 'component': 27, 'platform': 81, 'derives': 33, 'much': 70, 'synta
x': 106, 'from': 42, 'fewer': 40, 'low': 62, 'level': 58, 'facilities': 38, 'than': 108, 'either': 37, 'origi
nal': 77, 'reference': 92, 'compilers': 25, 'machines': 64, 'libraries': 59, 'under': 111, 'proprietary': 88,
'licenses': 61, 'may': 65, 'compliance': 26, 'specification': 101, 'community': 22, 'process': 86, 'relicense
d': 95, 'technologies': 107, 'gnu': 44, 'public': 89, 'license': 60, 'others': 79, 'have': 47, 'also': 2, 'al
ternative': 3, 'implementations': 50, 'such': 103, 'as': 10, 'compiler': 24, 'ytecode': 121, 'classpath': 19,
'standard': 102, 'icedtea': 48, 'browser': 14, 'plugin': 83, 'for': 41, 'applets': 6}]
```