AIR Assignment Dataset-1

Sreenath Saikumar PES2UG19CS406

Srivatsan Narendiran PES2UG19CS408

Sreekanth Maneesh PES2UG19CS405

Link to the Code used in the assignment:
https://github.com/SreenathSaikumar/AIR_assignment

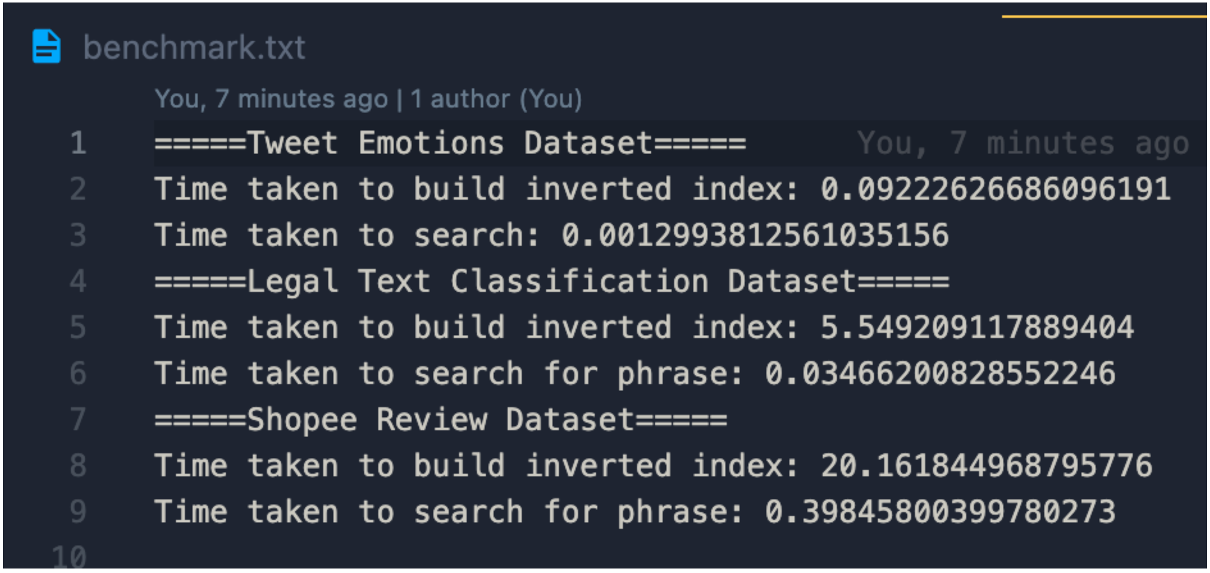
3 Datasets were chosen to be used for the assignment:

- Legal Citation Text Classification
- Shopee Text Reviews
- Emotion Detection from Text

The first 2 datasets were used to demonstrate and benchmark phrase querying with proximity while the 3rd dataset was used to perform simple Boolean queries with AND, OR and NOT.

Benchmarks:

Both Positional index creation or Inverted index creation in the case of the 3rd dataset as well as query retrieval times were benchmarked to infer an approximate performance comparison.



```
=====Tweet Emotions Dataset=====
Time taken to build inverted index: 0.09222626686096191
Time taken to search: 0.0012993812561035156
=====Legal Text Classification Dataset=====
Time taken to build inverted index: 5.549209117889404
Time taken to search for phrase: 0.03466200828552246
=====Shopee Review Dataset=====
Time taken to build inverted index: 20.161844968795776
Time taken to search for phrase: 0.39845800399780273
```

This file was generated while running the index construction and queries on all the datasets

Code for Dataset 1 (Legal Classification):

Phrase Querying with proximity has been performed

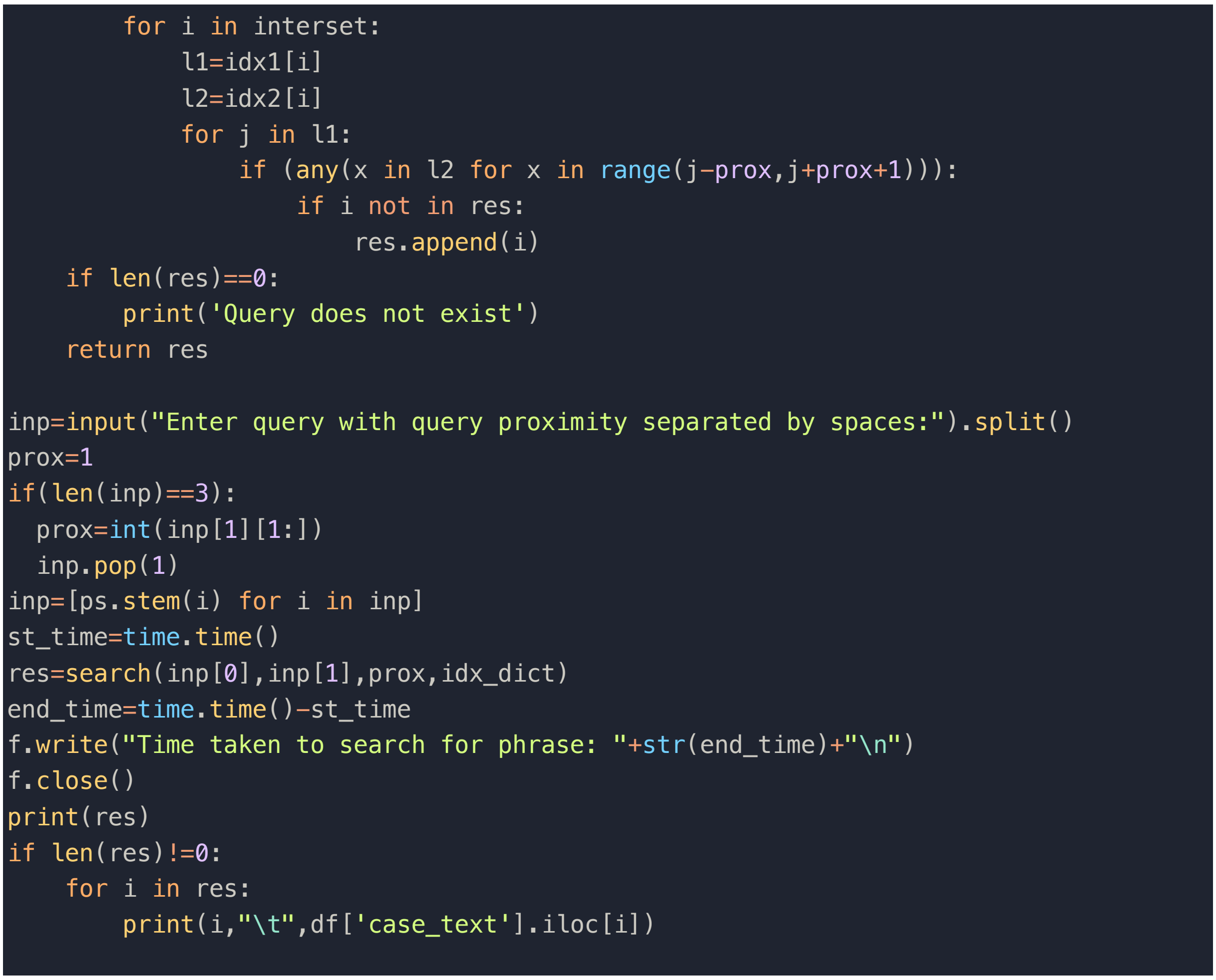
```
from nltk.tokenize import word_tokenize
from nltk.tokenize import sent_tokenize
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
import pandas as pd
import time

df=pd.read_csv('legal_text_classification.csv')
stop_words=set(stopwords.words('english'))
df=df.dropna()
df['texttoken']=df['case_text'].apply(word_tokenize)
df['texttoken']=df['texttoken'].apply(lambda words:[word.lower() for word in words
if word.isalpha()])
df['stop_remd']=df['texttoken'].apply(lambda x:[item for item in x if item not in
stop_words])

ps=PorterStemmer()
df['stemmed']=df['stop_remd'].apply(lambda x:[ps.stem(item) for item in x])


idx_dict={}
st_time=time.time()
for postext,text in enumerate(df['stemmed']):
    for pos,term in enumerate(text):
        if term not in idx_dict.keys():
            idx_dict[term]=[0,{}]
        idx_dict[term][0]+=1
        if postext not in idx_dict[term][1].keys():
            idx_dict[term][1][postext]=[]
        idx_dict[term][1][postext].append(pos)
end_time=time.time()-st_time
f=open("benchmark.txt",'a')
f.write("=====Legal Text Classification Dataset=====\n")
f.write("Time taken to build inverted index: "+str(end_time)+"\n")
print("Term [Frequency,Entry number:[Positions in that entry]]")
for i in list(idx_dict)[:10]:
    print(i,idx_dict[i])

def search(t1,t2,prox,idx_dict):
    res=[]
    if t1 not in idx_dict.keys() or t2 not in idx_dict.keys():
        print("Query does not exist")
    else:
        idx1=idx_dict[t1][1]
        idx2=idx_dict[t2][1]
        interset=set(idx1.keys()).intersection(idx2.keys())
```



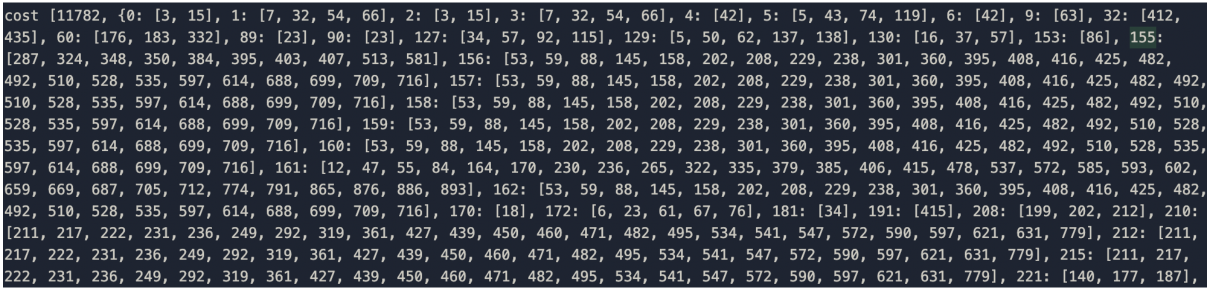
```
        for i in interset:
            l1=idx1[i]
            l2=idx2[i]
            for j in l1:
                if (any(x in l2 for x in range(j-prox,j+prox+1))):
                    if i not in res:
                        res.append(i)
    if len(res)==0:
        print('Query does not exist')
    return res

inp=input("Enter query with query proximity separated by spaces:").split()
prox=1
if(len(inp)==3):
  prox=int(inp[1][1:])
  inp.pop(1)
inp=[ps.stem(i) for i in inp]
st_time=time.time()
res=search(inp[0],inp[1],prox,idx_dict)
end_time=time.time()-st_time
f.write("Time taken to search for phrase: "+str(end_time)+"\n")
f.close()
print(res)
if len(res)!=0:
    for i in res:
        print(i,"\t",df['case_text'].iloc[i])
```

Outputs:

Positional index outputs are of the form ‘Term [Total Frequency, Entry number:[Positions in that entry]]’ in the output

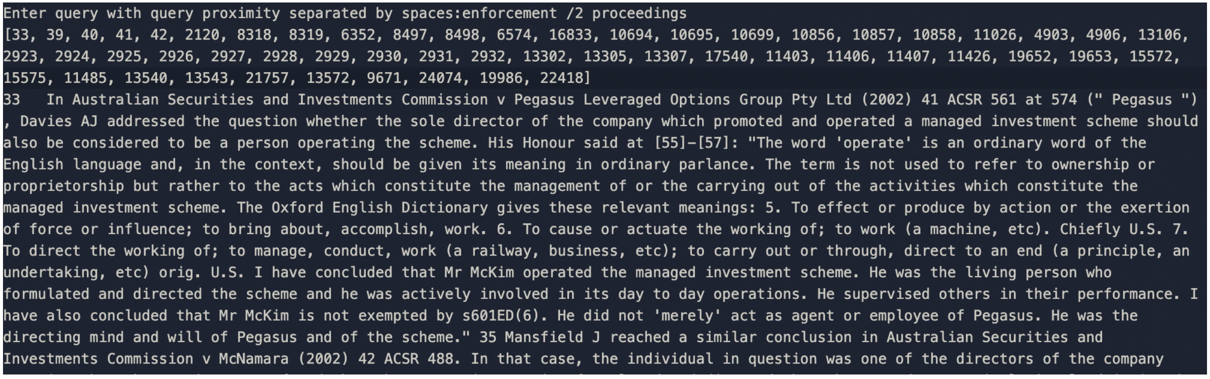
For example:



```
cost [11782, {0: [3, 15], 1: [7, 32, 54, 66], 2: [3, 15], 3: [7, 32, 54, 66], 4: [42], 5: [5, 43, 74, 119], 6: [42], 9: [63], 32: [412,
435], 60: [176, 183, 332], 89: [23], 90: [23], 127: [34, 57, 92, 115], 129: [5, 50, 62, 137, 138], 130: [16, 37, 57], 153: [86], 155:
[287, 324, 348, 350, 384, 395, 403, 407, 513, 581], 156: [53, 59, 88, 145, 158, 202, 208, 229, 238, 301, 360, 395, 408, 416, 425, 482,
492, 510, 528, 535, 597, 614, 688, 699, 709, 716], 157: [53, 59, 88, 145, 158, 202, 208, 229, 238, 301, 360, 395, 408, 416, 425, 482, 492,
510, 528, 535, 597, 614, 688, 699, 709, 716], 158: [53, 59, 88, 145, 158, 202, 208, 229, 238, 301, 360, 395, 408, 416, 425, 482, 492, 510,
528, 535, 597, 614, 688, 699, 709, 716], 159: [53, 59, 88, 145, 158, 202, 208, 229, 238, 301, 360, 395, 408, 416, 425, 482, 492, 510, 528,
535, 597, 614, 688, 699, 709, 716], 160: [53, 59, 88, 145, 158, 202, 208, 229, 238, 301, 360, 395, 408, 416, 425, 482, 492, 510, 528, 535,
597, 614, 688, 699, 709, 716], 161: [12, 47, 55, 84, 164, 170, 230, 236, 265, 322, 335, 379, 385, 406, 415, 478, 537, 572, 585, 593, 602,
659, 669, 687, 705, 712, 774, 791, 865, 876, 886, 893], 162: [53, 59, 88, 145, 158, 202, 208, 229, 238, 301, 360, 395, 408, 416, 425, 482,
492, 510, 528, 535, 597, 614, 688, 699, 709, 716], 170: [18], 172: [6, 23, 61, 67, 76], 181: [34], 191: [415], 208: [199, 202, 212], 210:
[211, 217, 222, 231, 236, 249, 292, 319, 361, 427, 439, 450, 460, 471, 482, 495, 534, 541, 547, 572, 590, 597, 621, 631, 779], 212: [211,
217, 222, 231, 236, 249, 292, 319, 361, 427, 439, 450, 460, 471, 482, 495, 534, 541, 547, 572, 590, 597, 621, 631, 779], 215: [211, 217,
222, 231, 236, 249, 292, 319, 361, 427, 439, 450, 460, 471, 482, 495, 534, 541, 547, 572, 590, 597, 621, 631, 779], 221: [140, 177, 187],
```

Snapshot of the positional index

The phrase query uses the positional index to query



```
Enter query with query proximity separated by spaces:enforcement /2 proceedings
[33, 39, 40, 41, 42, 2120, 8318, 8319, 6352, 8497, 8498, 6574, 16833, 10694, 10695, 10699, 10856, 10857, 10858, 11026, 4903, 4906, 13106,
2923, 2924, 2925, 2926, 2927, 2928, 2929, 2930, 2931, 2932, 13302, 13305, 13307, 17540, 11403, 11406, 11407, 11426, 19652, 19653, 15572,
15575, 11485, 13540, 13543, 21757, 13572, 9671, 24074, 19986, 22418]
33   In Australian Securities and Investments Commission v Pegasus Leveraged Options Group Pty Ltd (2002) 41 ACSR 561 at 574 (" Pegasus ")
, Davies AJ addressed the question whether the sole director of the company which promoted and operated a managed investment scheme should
also be considered to be a person operating the scheme. His Honour said at [55]-[57]: "The word 'operate' is an ordinary word of the
English language and, in the context, should be given its meaning in ordinary parlance. The term is not used to refer to ownership or
proprietorship but rather to the acts which constitute the management of or the carrying out of the activities which constitute the
managed investment scheme. The Oxford English Dictionary gives these relevant meanings: 5. To effect or produce by action or the exertion
of force or influence; to bring about, accomplish, work. 6. To cause or actuate the working of; to work (a machine, etc). Chiefly U.S. 7.
To direct the working of; to manage, conduct, work (a railway, business, etc); to carry out or through, direct to an end (a principle, an
undertaking, etc) orig. U.S. I have concluded that Mr McKim operated the managed investment scheme. He was the living person who
formulated and directed the scheme and he was actively involved in its day to day operations. He supervised others in their performance. I
have also concluded that Mr McKim is not exempted by s601ED(6). He did not 'merely' act as agent or employee of Pegasus. He was the
directing mind and will of Pegasus and of the scheme." 35 Mansfield J reached a similar conclusion in Australian Securities and
Investments Commission v McNamara (2002) 42 ACSR 488. In that case, the individual in question was one of the directors of the company
```

Snapshot of the output

The output of the phrase query is a list of all the matching documents followed by the contents of each document.