

AIR Assignment Dataset-2

Sapna Singh PES2UG19CS366

Siddharth Chattar PES2UG19CS388

Sneha Sujit Saha PES2UG19CS393

Link to the Code used in the assignment:

https://github.com/sapnasingh2041/AIWIR_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.

```
benchmark.txt
You, 7 minutes ago | 1 author (You)
1  =====Tweet Emotions Dataset===== You, 7 minutes ago
2  Time taken to build inverted index: 0.09222626686096191
3  Time taken to search: 0.0012993812561035156
4  =====Legal Text Classification Dataset=====
5  Time taken to build inverted index: 5.549209117889404
6  Time taken to search for phrase: 0.03466200828552246
7  =====Shopee Review Dataset=====
8  Time taken to build inverted index: 20.161844968795776
9  Time taken to search for phrase: 0.39845800399780273
10
```

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

Code for Dataset 2 (Shopee Reviews):

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('shopee_reviews.csv',usecols=["text"])
stop_words=set(stopwords.words('english')) df=df.dropna()
df['texttoken']=df['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("====Shopee Review 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['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:

```

Term [Frequency,Entry number:[Positions in that entry]]
look [104405, {0: [0], 2: [3], 6: [7], 35: [0], 46: [2], 48: [1], 54: [5], 71: [3], 87: [1], 104: [0], 110: [1], 116: [3], 170: [2], 191:
[20], 194: [0], 255: [1], 258: [8, 10], 268: [0], 301: [5], 317: [0], 320: [7], 341: [8], 347: [0], 404: [8], 427: [3], 472: [0], 489:
[11], 530: [0], 536: [4], 538: [1], 553: [5], 570: [1], 581: [0], 602: [5], 612: [2], 621: [2], 622: [5], 634: [0], 650: [9], 665: [4, 8],
676: [3], 679: [5, 10], 691: [5, 19], 694: [4], 706: [3], 708: [13], 734: [4], 738: [5, 11], 751: [5], 775: [4, 8], 797: [0], 798: [0],
809: [0], 819: [1], 824: [8], 833: [0], 837: [8], 848: [1], 854: [7], 868: [2], 873: [1], 903: [3, 14], 913: [11], 918: [0], 919: [6],
931: [0], 934: [0], 937: [5], 940: [5, 6], 970: [10], 989: [3], 1011: [9], 1015: [0], 1016: [11], 1022: [3], 1060: [7], 1068: [1], 1081:
[6], 1083: [2], 1095: [9], 1098: [4], 1110: [1], 1118: [1], 1122: [14], 1143: [13], 1159: [4], 1186: [20], 1192: [0], 1198: [0], 1213: [4],
1260: [3], 1287: [6], 1303: [5], 1319: [2], 1346: [2], 1347: [7], 1356: [1], 1362: [0], 1371: [3], 1393: [2], 1407: [0], 1411: [4],
1417: [11], 1418: [1], 1452: [1], 1462: [0, 3], 1519: [6], 1520: [3], 1543: [3], 1561: [4], 1563: [7], 1565: [2, 6, 15], 1569: [1], 1575:
[0], 1579: [0], 1585: [1], 1590: [0], 1591: [1], 1606: [1], 1614: [0], 1615: [0], 1627: [6], 1628: [6], 1629: [0], 1653: [3], 1717: [0],
1723: [4], 1735: [7], 1745: [4], 1779: [6], 1819: [0], 1833: [19], 1835: [12], 1842: [1], 1901: [4], 1908: [2], 1918: [5], 1937: [11],
2000: [8], 2003: [3], 2010: [5], 2020: [2], 2039: [3], 2130: [1], 2138: [16], 2151: [5], 2156: [0], 2167: [3], 2182: [11], 2185: [14],
2190: [2], 2193: [0], 2198: [9], 2199: [9], 2225: [4], 2233: [7], 2242: [6], 2294: [1], 2302: [3], 2307: [15], 2329: [0], 2330: [3],
2353: [5], 2355: [3], 2380: [0], 2384: [3], 2388: [1], 2405: [3], 2410: [3], 2422: [3], 2428: [0], 2444: [5], 2463: [3], 2469: [1],

```

Snapshot of the positional index

The phrase query uses the positional index to query

```

Enter query with query proximity separated by spaces:late /5 delivery
[1335299, 491531, 1089560, 917534, 1450030, 49212, 991296, 1441874, 1187924, 974934, 311383, 1187936, 409699, 295013, 843875, 917616,
1040498, 327805, 1278081, 1286275, 32906, 589969, 614561, 1417381, 327853, 270515, 1245370, 204990, 557247, 590013, 901326, 843993,
237790, 1196257, 958693, 418022, 860391, 729322, 467185, 516342, 516354, 98587, 524576, 1237280, 819491, 1376547, 57637, 819495, 819505,
401714, 1442102, 57664, 1483079, 262494, 942441, 762220, 999793, 631171, 1327494, 1261959, 532891, 663964, 336290, 836004, 893349,
106919, 1081770, 557486, 1311151, 1425847, 549307, 1180096, 1311175, 893385, 1311189, 565719, 1196506, 1475036, 483813, 1081836, 901613,
123377, 66035, 1253894, 1450511, 426518, 918040, 1352216, 623152, 983605, 1458745, 1147456, 975430, 1098312, 508491, 16974, 557655,
205401, 1090138, 934504, 1041003, 1409666, 1483396, 1057414, 1196683, 656013, 1131149, 746133, 901781, 819868, 1254048, 484002, 590498,
205485, 934583, 1426132, 33506, 25315, 1295074, 410350, 1295087, 434956, 557844, 1426199, 557850, 99111, 1098544, 1205041, 680762, 99135,
1352526, 1319770, 910181, 1344363, 893854, 533413, 525225, 287662, 58289, 467897, 1450941, 1450944, 107459, 418759, 926669, 140240, 9169,
926678, 893914, 115677, 508895, 1434592, 779248, 533493, 533496, 812030, 1295371, 1188877, 549921, 214053, 549930, 1254443, 525359,
492593, 1057860, 599109, 926792, 525391, 582735, 861267, 1360979, 1451091, 1049713, 926846, 435327, 926851, 1426563, 140428, 9358,
1434766, 91284, 525472, 1279139, 926887, 722094, 1057977, 1008827, 722112, 951488, 427204, 492760, 378086, 1393898, 632053, 1049847,
1311991, 812293, 402695, 435476, 1295643, 1189159, 558393, 517441, 812353, 1164609, 1008970, 427352, 886105, 910706, 828787, 1197431,
1484151, 1402233, 525690, 222631, 198067, 533939, 1189307, 1304001, 124354, 533962, 1197520, 239067, 1115612, 288220, 476635, 927196,
599524, 304615, 894441, 1476083, 1230328, 1230341, 525835, 230933, 894495, 1148450, 591401, 198187, 17964, 550455, 1197625, 1197626,
1197628, 894525, 362063, 1156718, 1189489, 1189490, 378491, 370302, 935556, 231060, 255649, 730785, 452264, 468655, 632505, 1025730,
771780, 607946, 378572, 919245, 911067, 485086, 886501, 329450, 395004, 108286, 542466, 1459981, 345872, 1337104, 247570, 821015,
1287961, 526108, 919330, 345891, 1460014, 542514, 902984, 1238865, 943970, 190318, 149361, 403323, 1320830, 42888, 558988, 1181601,
1468270 002707 50272 576777 010401 547661 354746 550050 550051 550052 003117 550071 116704 534407 1468287 550080 715010
1007579, 1114075, 344036, 1245160, 1122299, 532479]
1335299 A bit late delivery but markers are good and nice to use! Thank you.
491531 Delivery was so late and even got lost. Seller sent replacement within 3 days of informing. Thank you for sending a replacement
quickly.
1089560 Although not very pleased with the delivery and late response, seller was polite enough. Boxes came crushed and only for own
consumption, not as gift.
917534 Received the next day of order, but seller is kind to inform of late delivery. Yet to try but looks good.
1450030 So far so good. Late delivery may be due to circuit breaker.
49212 Late delivery ...item is good
991296 At first my delivery was failed and i noticed late. I contacted the seller/shopee and quickly had a reply. Next morning item
were delivered again. I may be outside when it was first came in my house. Pj is very pretty, i really love it! I will order again! 🍀
1441874 The book are well wrapped.. Tho a bit late on the ninja delivery side.. Seller response nicely and apologize for the delay.
. Such a nice gesture frm a seller.... ☺

```

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.