

CSCE 636 - Project part 5

Sanjeev Narayanan - UIN:226007203

April 14, 2019

1 Introduction

Semantic vector space models of language represent each word with a real-valued vector. These vectors can be used as features in a variety of applications, such as information retrieval, document classification, question answering, named entity recognition, and parsing. Traditionally, keyword research involved building a list or database of relevant keywords that we hoped to rank for. Often graded by difficulty score, click through rate and search volume, keyword research was about finding candidates in this list to create content and gather some organic traffic through exact matching. While this method of keyword research is still relevant, the landscape has changed.

This project deals with searching for a relevant article based on keywords used to look up for the article. Over the last few decades, scientific papers often come up with an section called “key-words” to help look up for those articles online. But the vast multitude of such documents makes it difficult to get to the right document. In order to go beyond search keywords, it’s important to build more meaningful keyword lists or databases that are rich in context and take the searchers intent into account. This project aims to build such a tool that will point to a right set of documents for the user on an offline interface.

2 Literature Survey

Text representation learning has been extensively studied. Popular representations range from the simplest Bag-of-Words method and its term-frequency based variants [1], language model based methods [2] [3], topic models [4] and distributed vector representations [5].

Word2Vec proposed a neural network architecture of an input layer, a projection layer parameterized by the matrix U and an output layer by V^T . It defines the probability of observing the target word w_t in a document D given its local context c^t as

$$P(w^t|c^t) = \frac{\exp(v_{w^t}^T U c^t)}{\sum_{w' \in V} \exp(v_{w'}^T U c^t)}$$

where:

$U \in \mathcal{R}^{h \times v}$: the projection matrix from the input space to a hidden space of size h . We use

u_w to denote the column in U for word w , i.e., the “input” vector of word w .
 $V^T \in \mathcal{R}^{v \times h}$: the projection matrix from the hidden space to output. Similarly, we use v_w to denote the column in V for word w , i.e., the “output” vector of word w
 $c^t \in \mathcal{R}^{v \times 1}$: Bag of words of the local context $w^{t-k}, \dots, w^{t-1}, w^{t+1}, \dots, w^{t+k}$ at the target position t . $c_j^t = 1$ if word j appears within the sliding window of the target.

The word vectors are then learned to maximize the log likelihood of observing the target word at each position of the document.

Paragraph Vectors, on the other hands, explicitly learns a document vector with the word embeddings. It introduces another projection matrix $D \in \mathcal{R}^{h \times n}$. Each column of D acts as a memory of the global topic of the corresponding document. It then defines the probability of observing the target word w^t in a document D given its local context c^t as

$$P(w^t | c^t, d) = \frac{\exp(v_{w^t}^T (U c^t + d))}{\sum_{w' \in V} \exp(v_{w'}^T (U c^t + d))}$$

where $d \in D$ is the vector representation of the document. As we can see from this formula, the complexity of Paragraph Vectors grows with not only the size of the vocabulary, but also the size of the training corpus.

In this model, we use word vectors obtained using GloVe vectors [6] to train the model and in order to verify our results, we use paragraph vectors.

3 Dataset

The dataset is a set of accepted papers from the Neural Information Processing System Conference, 2015 (NIPS 2015) ¹. The set contains about 400 Computer Vision and related Scientific papers available for free download from Kaggle,². The raw dataset needed a lot of screening and cleaning before it can be processed for the training.

- Convert each pdf into a text document using a pdfminer tool ^{3,4}
- Split each document into its title and body
- Strip the document of links, citations, stop words and other special characters.
- Tokenize the words in the document
- The dataset is now ready to be processed for training.

¹<https://nips.cc/Conferences/2015>

²<https://www.kaggle.com/benhamner/nips-2015-papers/version/2/home>

³Convert each pdf into a text document using a pdfminer tool

⁴<https://stackoverflow.com/questions/26494211/extracting-text-from-a-pdf-file-using-pdfminer-in-python>

4 Neural Network Model

The neural network architecture is based on an LSTM followed by a Dense layer.

- 1 LSTM layer with 256 cells
- 20% dropout
- Dense layer of size 1 to output the title of the document.

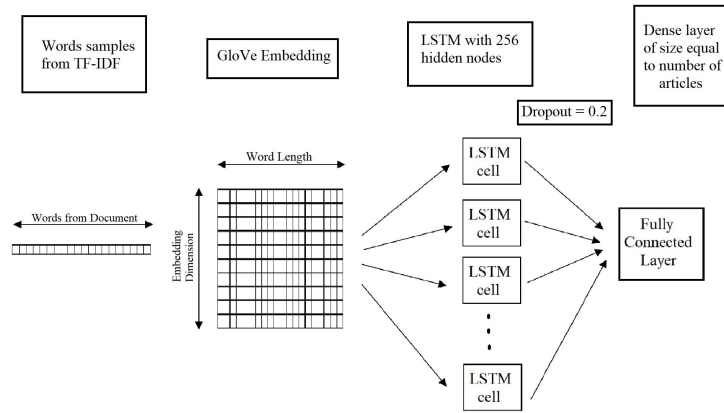


Figure 1: Network Architecture

5 Training and Performance

5.1 Old-Fashioned Inverted Index Method

The complete data set was split into individual arrays and all the unique words in the data were computed. Each title was indexed to a number and each number was indexed to the title. The glove vectors were generated by training on the dataset. This way each keyword was vectorized (including proper nouns). A user input tool accepts keywords, performs an autocorrection on each keyword and confirms the intended meaning of each keyword input to extend the semantic understanding. The model then takes in 5 most similar keywords and searches through the data with this extended keyword set and obtains a list of titles for each word in the extended set. The intersection of the titles for each keyword gives the possible top 5 or 10 most related titles with the keywords input. In place of training the glove vectors on the dataset, the pre-trained vectors can also be employed. Since, we require for the search to be hard mined and produce words that are most closely associated with the keyword input, glove vectors are generated for this search method. The results of this method are not surprising as it has time and again proved to be fairly accurate.

5.2 LSTM based Approach

An LSTM followed by a dense layer is employed in this method. Instead of providing a user input keyword search, for initial understanding, the model was trained on a few sentences to test the accuracy. Random sentences with at least 30 words were taken and the last 5 words were pretended to be the title (output). The input was the entire sentence initially, but further tests on reduced input length with 5 - 6 words gave nearly error-less results. In the next phase, the entire dataset used in the previous case was taken with the key- word input fixed to 4. Each word was converted to its glove vector. If in case a random word chosen did not have a corresponding vector, the algorithm searched for another word until the chosen word with vector is found. This logic was extended till the user can input a continuous stream of keys in the form as a sentence.

The model was trained on an NVIDIA 2 Tesla card GPU for 100 epochs. The training improved to an accuracy of 0.8977 with a 0.32 loss. The loss is high due to the fact that it is possible that the randomly selected keyword may be present in multiple files. This was verified with a test case example, where a 4 random words were selected from a file. The model precisely located “a file” where the chosen keywords were present, but did not accurately point to the true output. However, when we consider the probabilities of output files, the prediction covers the true output within the top 5 output values (probabilities sorted in descending order). The whole sequence is as follows:

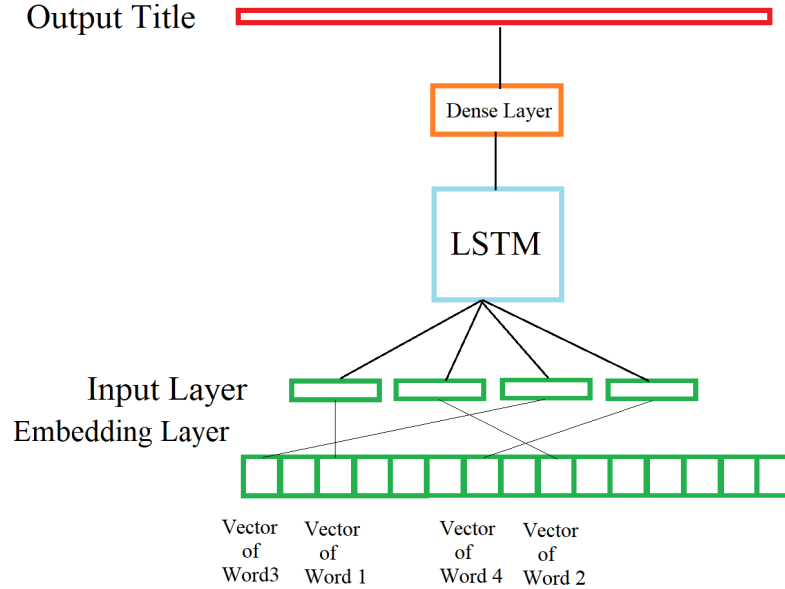


Figure 2: Results Window

6 Running the GUI

A GUI was made on tkinter to view how the model works from an user's perspective. The GUI has 2 parts. The first one is to test the model and the second one is to verify that our results are right.

The Youtube link to the demo is given here : <https://youtu.be/jgrfciDYHd4>

You will need the following packages to run the GUI

```
1 nltk
2 gensim
3 glove
4 tkinter
```

Download the folder from the Github link and navigate to the main project directory. Find the python gui file and run

```
1 python gui-updated.py
```

Once the gui opens, you will see a pop up window with the following.

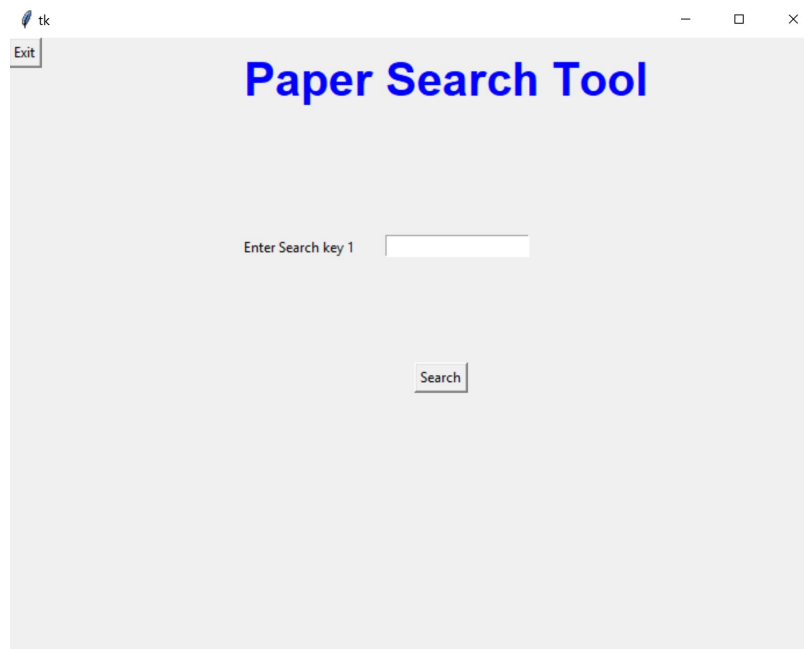


Figure 3: Main Window

Go ahead and enter the search query. Since this program is designed to recognize those words which are a part of documents in NIPS 2015, common words like “hello”, “boy”, etc may not be recognized. The model will return an error highlighting you what to edit. Once the arguments are right, it will open another window with a list of top 10 papers related to the search query. The following image shows search results for the keyword “RNN”

Here, the user is provided with 2 options:



Figure 4: Results Window

- The user can view each of the papers under the “View Papers” option

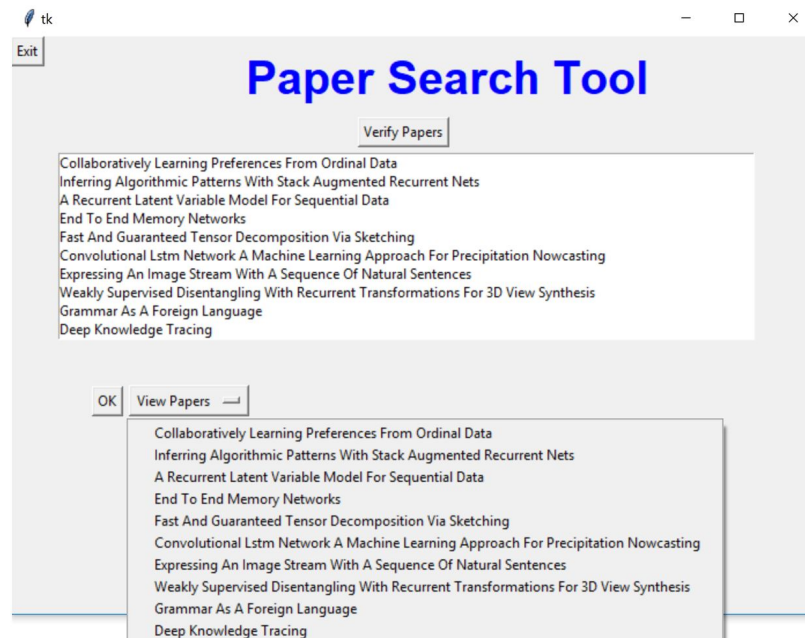


Figure 5: View Papers

- The user can verify his results by advancing his search queries using the “Verify Papers” option. Here, the user is allowed to choose only different papers to narrow his search option.

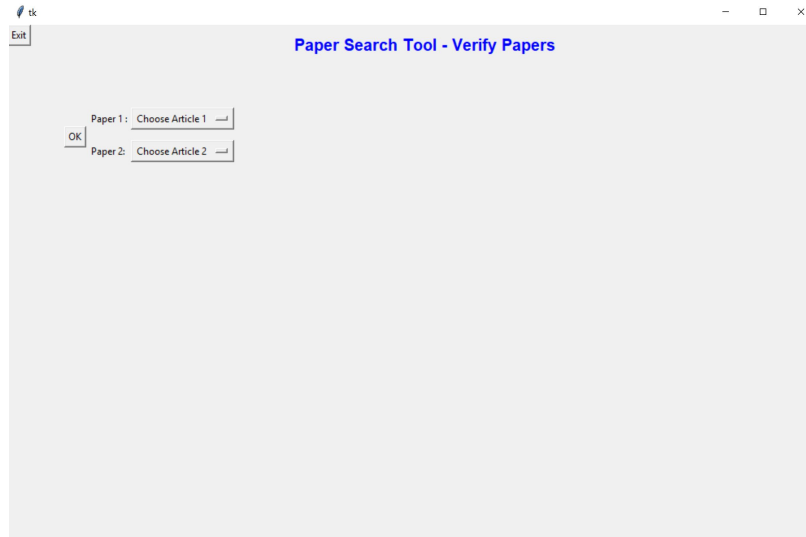


Figure 6: Verify Papers (I)

On selection, two windows with the 2 papers open up with a text box to enter his next input show up. Once the user hits “Predict”, the model displays which paper the results came from. This approach is based on the cosine similarity.

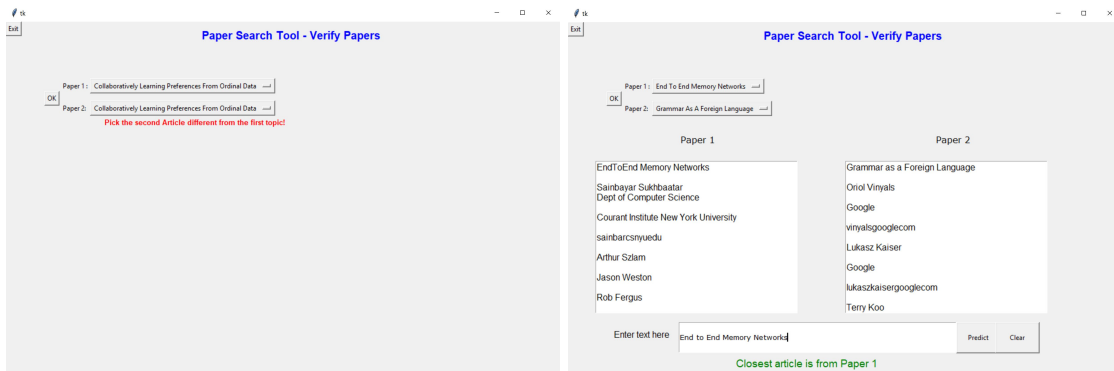


Figure 7: Verify Papers (II)

7 Conclusion and Future Work

This project successfully developed a tool that returns a set of highly probable articles, given the keywords present in it. We aim to capture the semantic cognition of the keywords and understand what the user is exactly looking for. The applications related to this are innumerable. This model can be trained on other datasets. A history of records related to law, proceedings, medical data are a few obvious mentions. The respective user is aided in drawing references very easily by using this tool as a look up contraption. This model used an LSTM network with a very few layers. While we understand that a certain trade-off can be made between computation and accuracy, one can easily modify this architecture when presented with larger datasets and more intricate nuances to search for better accuracy. Also this model can be tweaked further to ensure that it is noise tolerant. Noise in this context can mean those words in a query that can be linked to a particular article, but are not the exact words in the article.

References

- [1] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.
- [2] W Bruce Croft and John Lafferty. *Language modeling for information retrieval*, volume 13. Springer Science & Business Media, 2013.
- [3] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černocký, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*, 2010.
- [4] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.
- [5] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196, 2014.
- [6] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

8 Annotated Codes

8.1 Training

```
1 import pickle
2 #Load the training data
3 f = open('X_data.pkl', 'rb')
```



```

4 X_data = pickle.load(f)
5 f.close()
6 g = open('Y_data_new_final.pkl', 'rb')
7 Y_data = pickle.load(g)
8 f.close()
9
10 #Load the required libraries
11 from keras.models import Sequential
12 from keras.layers import Dense, Dropout, LSTM
13 from sklearn.utils import shuffle
14
15 X, y = shuffle(X_data, Y_data, random_state=0)
16 #create the model
17 model = Sequential()
18 model.add(LSTM(256, input_shape=(X_data.shape[1], X_data.shape[2])))
19 model.add(Dropout(0.2))
20 model.add(Dense(Y_data.shape[1], activation='softmax'))
21 model.compile(loss='categorical_crossentropy', optimizer='adam')
22 print(model.summary())
23
24 model.fit(X, y, nb_epoch=100, batch_size=10)
25 model.save("my_model100_fin.h5")

```

8.2 GUI

```

1 import tkinter as tk
2 from tkinter import *
3 from tkinter import ttk
4 from gensim.models.doc2vec import Doc2Vec
5 from nltk.tokenize import word_tokenize
6 import os, re
7 import string
8 from scipy.spatial.distance import cosine
9
10 #Edit this path to the "d2v.model" file
11 path1 = "C:/Users/Sanjeev Narayanan/Desktop/Sem 3/Neural Networks/Project /
Updated GUI files/"
12 model = Doc2Vec.load(path1+"d2v.model")
13
14 #Edit this path to the "outs" folder
15 path = "C:/Users/Sanjeev Narayanan/Desktop/Sem 3/Neural Networks/Project /
Updated GUI files/outs/"
16 path3 = "C:/Users/Sanjeev Narayanan/Desktop/Sem 3/Neural Networks/Project /
Updated GUI files/pdfs/"
17
18 #Edit this path to the "glove.model" file
19
20 path2 = "C:/Users/Sanjeev Narayanan/Desktop/Sem 3/Neural Networks/Project /
Updated GUI files/"
21
22 from glove import Glove
23 glove = Glove.load(path2+'glove.model')

```

```

24 #####
25
26 filenames = []
27 all_files = []
28 title = []
29 for i in os.listdir(path):
30     filenames.append(path + '%s' %i)
31     with open(path+'%s' %i, 'r', encoding='utf-8') as myfile:
32         data = myfile.read()
33         data = re.sub(r'([^\s\w]|-)+', ' ', data)
34         data = "".join(filter(lambda char: char in string.printable, data))
35         all_files.append(data)
36         title.append(i.replace('-', ' ')[5:].title().split('.')[0])
37
38 file2 = []
39 for i in os.listdir(path3):
40     file2.append(path3 + '%s' %i)
41
42
43 class LoginPage():
44     def __init__(self):
45         self.root=tk.Tk()
46         self.root.geometry("700x500+200+250")
47         label = tk.Label(self.root, text="Paper Search Tool", fg="blue", font=(
48             "Arial Bold", 30))
49         label.place(x=200,y=10)
50
51         label_1 = tk.Label(self.root, text="Enter Search key 1")
52         self.entry_1 = tk.Entry(self.root)
53         label_1.place(x=200,y=170)
54         self.entry_1.place(x=325,y=170)
55         logbtn = tk.Button(self.root, text="Search", command = self.
56             _login_btn_clickked)
57         logbtn.place(x=350,y=280)
58         myButton = tk.Button(self.root, text="Exit", command = self.buttonPushed)
59         myButton.grid(row=10)
60
61         self.root.mainloop()
62
63     def buttonPushed(self):
64         self.root.destroy()
65
66     def _login_btn_clickked(self):
67         #print("Clicked")
68         self.search1 = self.entry_1.get()
69
70         list1 = word_tokenize(self.search1.lower())
71         if len(self.search1)==0:
72             label4 = tk.Label(self.root, text="Cannot leave any field blank!", fg
73                 ="red", font=("Arial Bold", 15))
74             label4.place(x=200,y=320)

```

```

73         elif len(self.search1)!=0:
74             for item in range(len(list1)):
75                 try:
76                     glove.most_similar(list1[item])
77                     label4 = tk.Label(self.root, text="
78
79                     ", font=("Arial Bold", 15))
80                     label4.place(x=200,y=340+(40*item))
81
82                     except Exception:
83                         print("works, Word %s is invalid! Please edit" %(list1[item]
84
85                         )))
86                         label4s = tk.Label(self.root, text="Word %s is invalid!
87                         Please edit" %(list1[item]), fg="red", font=("Arial Bold", 15))
88                         label4s.place(x=200,y=340+(40*item))
89
90                     temp = []
91                     if len(self.search1)!=0:
92                         for item in range(len(list1)):
93                             if glove.most_similar(list1[item]):
94                                 temp.append(list1[item])
95
96                     if temp == list1:
97                         label4 = tk.Label(self.root, text="
98
99                         ", font=("Arial Bold", 15))
100                         label4.place(x=200,y=320)
101                         RP = resultspage(self.search1)
102
103                     else:
104                         pass
105
106                     else:
107                         label4 = tk.Label(self.root, text="
108
109                         ", font=("Arial Bold", 15))
110                         label4.place(x=200,y=320)
111                         RP = resultspage(self.search1)
112
113
114
115 class resultspage():
116
117     def __init__(self, list1):
118         self.root=tk.Tk()
119         self.root.geometry("700x500+200+250")
120         self.choice1 = 11
121         self.var1 = StringVar()
122         label = tk.Label(self.root, text="Paper Search Tool", fg="blue", font=(
123
124         "Arial Bold", 30))
125         label.place(x=200,y=10)
126         myButton = tk.Button(self.root, text="Exit", command = self.
127         buttonPushed)
128         myButton.grid(row=10)

```

```

116         scrollbar = ttk.Scrollbar(self.root)
117         listbox = tk.Listbox(self.root,width = 100)
118         listbox.place(x=40,y=100)
119         list2= word_tokenize(list1.lower())
120         v1 = model.infer_vector(list2 , steps=20, alpha=0.025)
121         similar_doc = model.docvecs.most_similar(positive=[v1])
122
123
124 #         myButton2 = tk.Button(self.root , text="View Papers",command = self.
viewpapers)
125 #         myButton2.place(x=200,y=300)
126
127         myButton3 = tk.Button(self.root , text="Verify Papers",command = self.
verify)
128         myButton3.place(x=300,y=70)
129
130
131         out_docs = []
132         for i in similar_doc:
133             out_docs.append(title[int(i[0])])
134
135         for i in range(len(out_docs)):
136             listbox.insert(END, "%s"%(out_docs[i]))
137
138         self.outer = out_docs
139         listbox.config(yscrollcommand=scrollbar.set)
140         scrollbar.config(command=listbox.yview)
141
142         variable = StringVar(self.root)
143         variable.set("View Papers") # default value
144         w = OptionMenu(self.root, variable, "%s"%(out_docs[0]), "%s"%(out_docs
[1]), "%s"%(out_docs[2]), "%s"%(out_docs[3]), "%s"%(out_docs[4]), "%s"%(
out_docs[5]), "%s"%(out_docs[6]), "%s"%(out_docs[7]), "%s"%(out_docs[8]), "%
s"%(out_docs[9]))
145         w.place(x=100,y=300)
146
147         def ok():
148             tle = variable.get()
149             for i in range(len(title)):
150                 if title[i]==tle:
151                     val = i
152             from os import startfile
153             startfile(file2[val])
154
155         button = Button(self.root , text="OK" , command=ok)
156         button.place(x=70,y=303)
157
158
159
160         self.root.mainloop()
161
162     def buttonPushed(self):

```

```

163         self.root.destroy()
164
165     def verify(self):
166         VP=verify_paps(self.outer)
167
168
169
170 class verify_paps():
171     def __init__(self,out_docs):
172         self.root=tk.Tk()
173         self.root.geometry("1000x650+100+150")
174         label = tk.Label(self.root, text="Paper Search Tool – Verify Papers",
175 fg="blue", font=("Arial Bold", 15))
176         label.place(x=350,y=10)
177         myButton = tk.Button(self.root, text="Exit",command = self.
178 buttonPushed)
179         myButton.grid(row=10)
180         variable1 = StringVar(self.root)
181         variable1.set("Choose Article 1")
182         variable2 = StringVar(self.root)
183         variable2.set("Choose Article 2")# default value
184         label_1 = tk.Label(self.root, text="Paper 1 :")
185         label_1.place(x=100,y=105)
186         label_2 = tk.Label(self.root, text="Paper 2:")
187         label_2.place(x=100,y=145)
188         w1 = OptionMenu(self.root, variable1, "%s"%(out_docs[0]), "%s"%(
189 out_docs[1]), "%s"%(out_docs[2]), "%s"%(out_docs[3]), "%s"%(out_docs[4]), "
190 %s"%(out_docs[5]), "%s"%(out_docs[6]), "%s"%(out_docs[7]), "%s"%(out_docs
191 [8]), "%s"%(out_docs[9]))
192         w1.place(x=150,y=100)
193         w2 = OptionMenu(self.root, variable2, "%s"%(out_docs[0]), "%s"%(
194 out_docs[1]), "%s"%(out_docs[2]), "%s"%(out_docs[3]), "%s"%(out_docs[4]), "
195 %s"%(out_docs[5]), "%s"%(out_docs[6]), "%s"%(out_docs[7]), "%s"%(out_docs
196 [8]), "%s"%(out_docs[9]))
197         w2.place(x=150,y=140)
198         def ok():
199             tle1 = variable1.get()
200             tle2 = variable2.get()
201             for i in range(len(title)):
202                 if title[i]==tle1:
203                     val1 = i
204             for i in range(len(title)):
205                 if title[i]==tle2:
206                     val2 = i
207             print(val1, val2)
208             if val1==val2:
209                 label = tk.Label(self.root, text="Pick the second Article
210 different from the first topic!", fg="red", font=("Arial Bold", 10))
211                 label.place(x=175,y=170)
212             else:
213                 label = tk.Label(self.root, text="
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

```

205         "red", font=("Arial Bold", 10))
206         label.place(x=170,y=170)
207
208         label100 = tk.Label(self.root, text="Paper 1", font=("Verdana"
209         , 12))
210         label100.place(x=200,y=200)
211         label200 = tk.Label(self.root, text="Paper 2", font=("Verdana"
212         , 12))
213         label200.place(x=660,y=200)
214         T1 = tk.Text(self.root, height = 15, width = 40, font=("
215         Helvetica", 12))
216         T1.place(x=50,y=250)
217         self.quote1 = all_files[val1]
218         # self.stringfiles1 = [" ".join([l for l in self.quote1])]
219         T1.insert(END, self.quote1)
220         T1.config(state=DISABLED)
221
222         T2 = tk.Text(self.root, height = 15, width = 40, font=("
223         Helvetica", 12))
224         T2.place(x=500,y=250)
225         self.quote2 = all_files[val2]
226         # self.stringfiles2 = [" ".join([l for l in self.quote2])]
227         T2.insert(END, self.quote2)
228         T2.config(state=DISABLED)
229
230         label_1 = tk.Label(self.root, text="Enter text here", font=("
231         Helvetica", 12))
232         self.entry_1 = tk.Entry(self.root, font=("Verdana", 10))
233         label_1.place(x=80,y=550)
234         self.entry_1.place(x=200,y=540,width=500,height = 55)
235
236         logbtn = tk.Button(self.root, text="Predict", command = self.
237         _login_btn_clickked, height = 3,width = 10)
238         logbtn.place(x=700,y=540)
239         logbtn2 = tk.Button(self.root, text="Clear", command = self.
240         buttonPushed, height = 3,width = 10)
241         logbtn2.place(x=770,y=540)
242
243         button = Button(self.root, text="OK", command=ok)
244         button.place(x=70,y=125)
245         self.root.mainloop()
246         def buttonPushed(self):
247             self.root.destroy()
248         def _login_btn_clickked(self):
249             self.search1 = self.entry_1.get()
250             if len(self.search1)>0:
251
252                 self.model1 = model
253                 self.model2 = model
254
255                 self.test_data = word_tokenize(self.search1.lower())

```

```

249         v1 = self.model1.infer_vector(self.test_data, steps=20, alpha
=0.025)
250         v2 = self.model2.infer_vector(self.test_data, steps=20, alpha
=0.025)
251         v01 = self.model1.infer_vector(self.quote1, steps=20, alpha=0.025)
252         v02 = self.model2.infer_vector(self.quote2, steps=20, alpha=0.025)
253         distance1 = cosine(v1,v01)
254         distance2 = cosine(v2,v02)
255         print(distance1,distance2)
256         print(self.search1,"@#$@#$")
257
258         if distance1>distance2:
259             label300 = tk.Label(self.root, text="
", font=("Helvetica", 14))
260             label300.place(x=300,y=600)
261             label300 = tk.Label(self.root, text="Closest article is from
Paper 1",fg="green", font=("Helvetica", 14))
262             label300.place(x=300,y=600)
263         else:
264             label300 = tk.Label(self.root, text="
", font=("Helvetica", 14))
265             label300.place(x=300,y=600)
266             label300 = tk.Label(self.root, text="Closest article is from
Paper 2",fg="green", font=("Helvetica", 14))
267             label300.place(x=300,y=600)
268
269         else:
270             pass
271
272
273 LP=LoginPage()

```