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 offine 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

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

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 $w^{t-k}, ..., w^{t-1}, w^{t+1}, ..., w^{t+k}$ 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

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

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
[3]Convert 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.

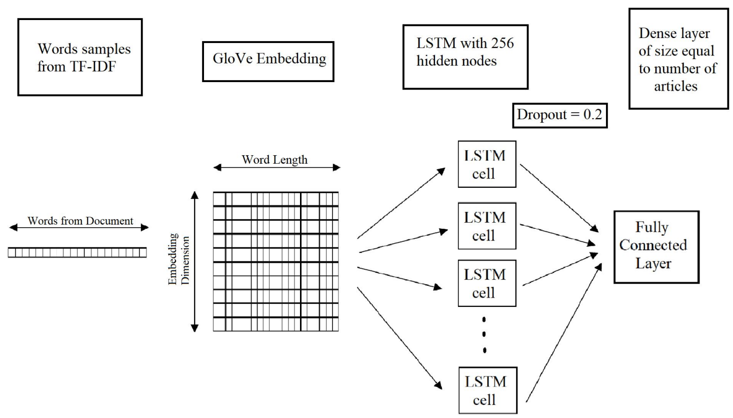

Figure 1: Network Architecture

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:

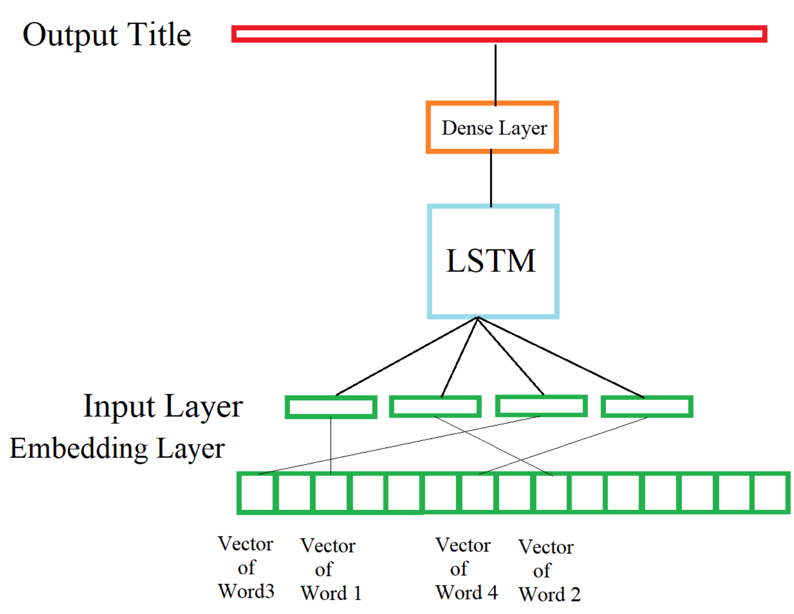

Figure 2: Results Window

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

```
nltk
gensim
glove
tkinter
```

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

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

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

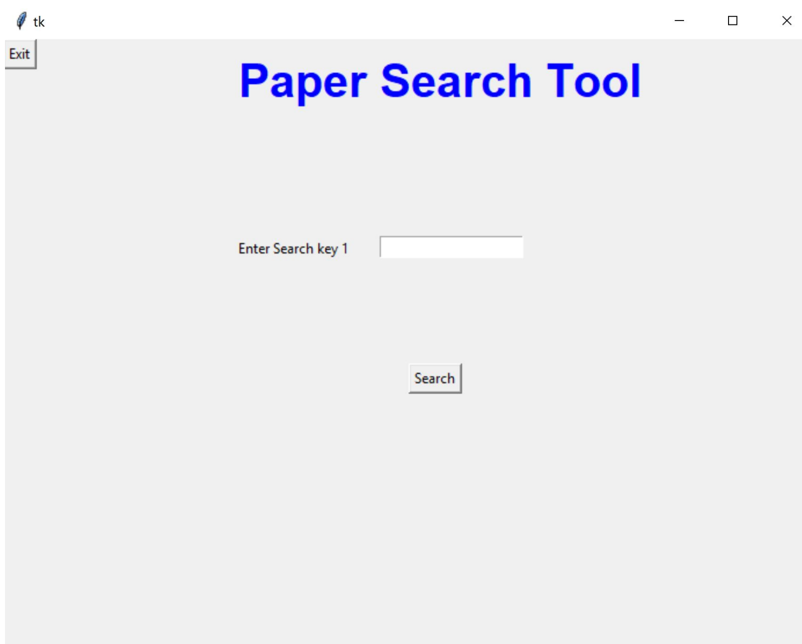

Figure 3: Main Window

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:

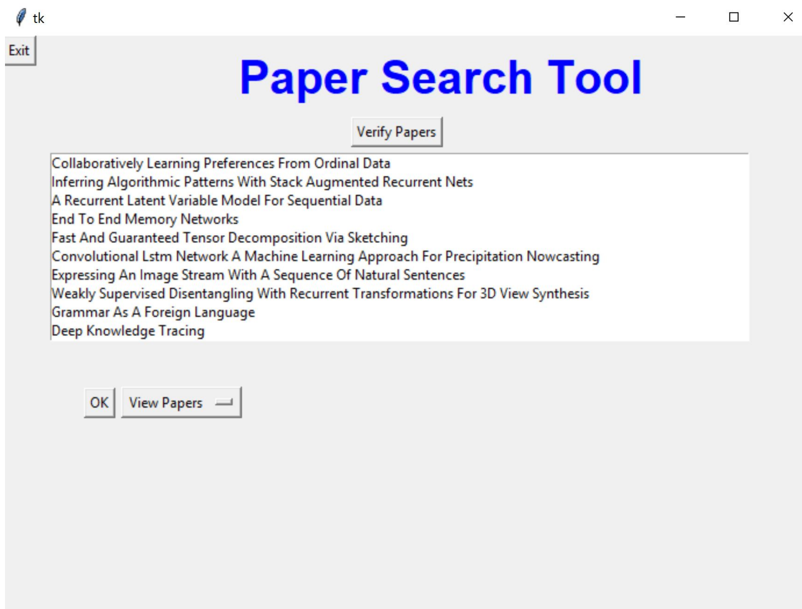

Figure 4: Results Window

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

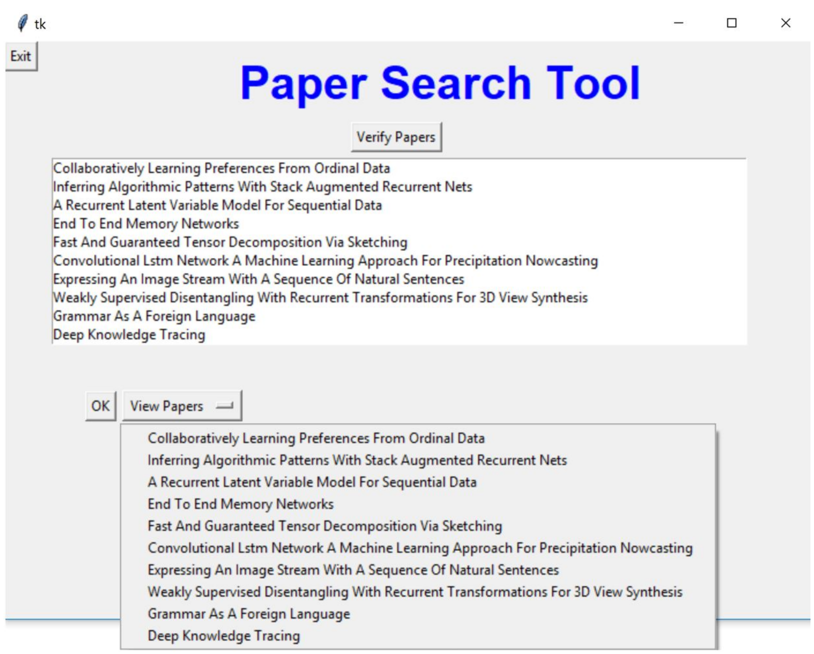

Figure 5: View Papers

- 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.

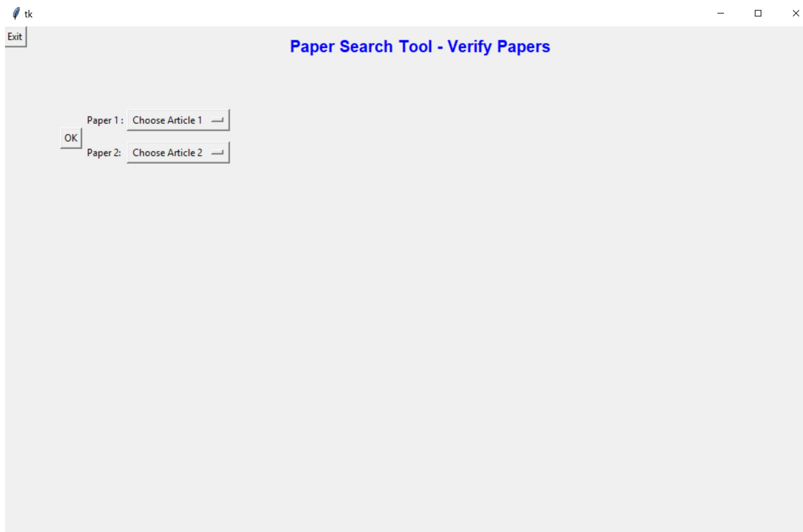

Figure 6: Verify Papers (I)

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.

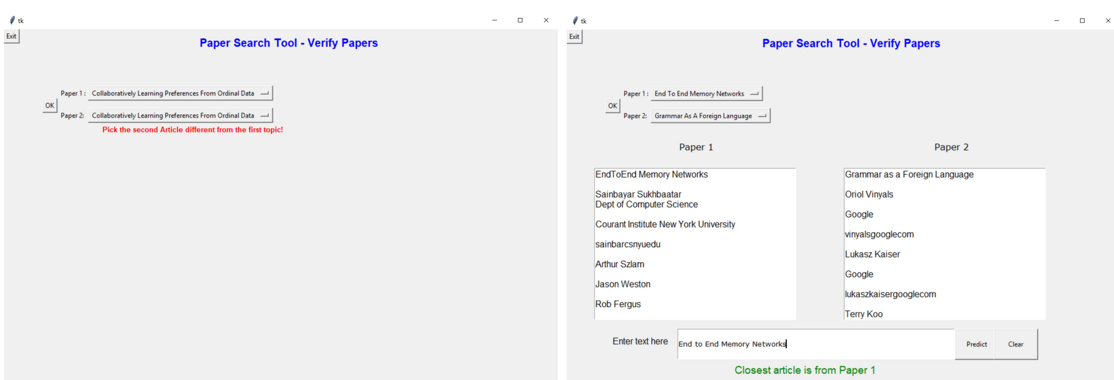

Figure 7: Verify Papers (II)

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 Černockỳ, 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

```
import pickle
#Load the training data
f = open('X_data.pckl', 'rb')
```

```
X_data = pickle.load(f)
f.close()
g = open('Y_data_new_final.pckl', 'rb')
Y_data = pickle.load(g)
f.close()

#Load the required libraries
from keras.models import Sequential
from keras.layers import Dense, Dropout, LSTM
from sklearn.utils import shuffle

X, y = shuffle(X_data, Y_data, random_state=0)
#create the model
model = Sequential()
model.add(LSTM(256, input_shape=(X_data.shape[1], X_data.shape[2])))
model.add(Dropout(0.2))
model.add(Dense(Y_data.shape[1], activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam')
print (model.summary())

model.fit(X, y, nb_epoch=100, batch_size=10)
model.save("my_model100_fin.h5")
```

8.2 GUI

```
import tkinter as tk
from tkinter import *
from tkinter import ttk
from gensim.models.doc2vec import Doc2Vec
from nltk.tokenize import word_tokenize
import os,re
import string
from scipy.spatial.distance import cosine

#Edit this path to the "d2v.model" file
path1 = "C:/Users/Sanjeev Narayanan/Desktop/Sem 3/Neural Networks/Project/
    Updated GUI files/"
model = Doc2Vec.load(path1+"d2v.model")

#Edit this path to the "outs" folder
path = "C:/Users/Sanjeev Narayanan/Desktop/Sem 3/Neural Networks/Project/
    Updated GUI files/outs/"
path3 = "C:/Users/Sanjeev Narayanan/Desktop/Sem 3/Neural Networks/Project/
    Updated GUI files/pdfs/"

#Edit this path to the "glove.model" file

path2 = "C:/Users/Sanjeev Narayanan/Desktop/Sem 3/Neural Networks/Project/
    Updated GUI files/"

from glove import Glove
glove = Glove.load(path2+'glove.model')
```

```python
##############################

filenames = []
all_files = []
title = []
for i in os.listdir(path):
    filenames.append(path + '%s' %i)
    with open(path+'%s' %i,'r',encoding='utf-8') as myfile:
        data = myfile.read()
    data = re.sub(r'([^\s\w]|_)+', '', data)
    data = "".join(filter(lambda char: char in string.printable, data))
    all_files.append(data)
    title.append(i.replace('-',' ')[5:].title().split('.')[0])

file2 = []
for i in os.listdir(path3):
    file2.append(path3 + '%s' %i)


class LoginPage():
   def __init__(self):
      self.root=tk.Tk()
      self.root.geometry("700x500+200+250")
      label = tk.Label(self.root, text="Paper Search Tool",fg="blue", font=("Arial Bold", 30))
      label.place(x=200,y=10)

      label_1 = tk.Label(self.root, text="Enter Search key 1")
      self.entry_1 = tk.Entry(self.root)
      label_1.place(x=200,y=170)
      self.entry_1.place(x=325,y=170)
      logbtn = tk.Button(self.root, text="Search", command = self._login_btn_clickked)
      logbtn.place(x=350,y=280)
      myButton = tk.Button(self.root, text="Exit",command = self.buttonPushed)
      myButton.grid(row=10)

      self.root.mainloop()

   def buttonPushed(self):
      self.root.destroy()

   def _login_btn_clickked(self):
      #print("Clicked")
      self.search1 = self.entry_1.get()

      list1 = word_tokenize(self.search1.lower())
      if len(self.search1)==0:
          label4 = tk.Label(self.root, text="Cannot leave any field blank!",fg="red", font=("Arial Bold", 15))
          label4.place(x=200,y=320)

```

```
      elif len(self.search1)!=0:
          for item in range(len(list1)):
              try:
                  glove.most_similar(list1[item])
                  label4 = tk.Label(self.root, text="
        ", font=("Arial Bold", 15))
                  label4.place(x=200,y=340+(40*item))

              except Exception:
                  print("works, Word %s is invalid! Please edit" %(list1[item]))
                  label4s = tk.Label(self.root, text="Word %s is invalid! Please edit" %(list1[item]) ,fg="red", font=("Arial Bold", 15))
                  label4s.place(x=200,y=340+(40*item))

          temp = []
          if len(self.search1)!=0:
              for item in range(len(list1)):
                  if glove.most_similar(list1[item]):
                      temp.append(list1[item])

              if temp == list1:
                  label4 = tk.Label(self.root, text="
        ", font=("Arial Bold", 15))
                  label4.place(x=200,y=320)
                  RP = resultspage(self.search1)

          else:
              pass

      else:
          label4 = tk.Label(self.root, text="
                          ", font=("Arial Bold", 15))
          label4.place(x=200,y=320)
          RP = resultspage(self.search1)


class resultspage():

    def __init__(self,list1):
        self.root=tk.Tk()
        self.root.geometry("700x500+200+250")
        self.choice1 = 11
        self.var1 = StringVar()
        label = tk.Label(self.root, text="Paper Search Tool",fg="blue", font=("Arial Bold", 30))
        label.place(x=200,y=10)
        myButton = tk.Button(self.root, text="Exit",command = self.buttonPushed)
        myButton.grid(row=10)
```

```
        scrollbar = ttk.Scrollbar(self.root)
        listbox = tk.Listbox(self.root,width = 100)
        listbox.place(x=40,y=100)
        list2= word_tokenize(list1.lower())
        v1 = model.infer_vector(list2, steps=20, alpha=0.025)
        similar_doc = model.docvecs.most_similar(positive=[v1])


#        myButton2 = tk.Button(self.root, text="View Papers",command = self.viewpapers)
#        myButton2.place(x=200,y=300)

        myButton3 = tk.Button(self.root, text="Verify Papers",command = self.verify)
        myButton3.place(x=300,y=70)


        out_docs = []
        for i in similar_doc:
            out_docs.append(title[int(i[0])])

        for i in range(len(out_docs)):
            listbox.insert(END, "%s" %(out_docs[i]))

        self.outer = out_docs
        listbox.config(yscrollcommand=scrollbar.set)
        scrollbar.config(command=listbox.yview)

        variable = StringVar(self.root)
        variable.set("View Papers") # default value
        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]))
        w.place(x=100,y=300)

        def ok():
            tle = variable.get()
            for i in range(len(title)):
                if title[i]==tle:
                    val = i
            from os import startfile
            startfile(file2[val])

        button = Button(self.root, text="OK", command=ok)
        button.place(x=70,y=303)



        self.root.mainloop()

    def buttonPushed(self):
```

```
        self.root.destroy()

    def verify(self):
        VP=verify_paps(self.outer)



class verify_paps():
    def __init__(self,out_docs):
        self.root=tk.Tk()
        self.root.geometry("1000x650+100+150")
        label = tk.Label(self.root, text="Paper Search Tool - Verify Papers", fg="blue", font=("Arial Bold", 15))
        label.place(x=350,y=10)
        myButton = tk.Button(self.root, text="Exit",command = self.buttonPushed)
        myButton.grid(row=10)
        variable1 = StringVar(self.root)
        variable1.set("Choose Article 1")
        variable2 = StringVar(self.root)
        variable2.set("Choose Article 2")# default value
        label_1 = tk.Label(self.root, text="Paper 1 :")
        label_1.place(x=100,y=105)
        label_2 = tk.Label(self.root, text="Paper 2:")
        label_2.place(x=100,y=145)
        w1 = OptionMenu(self.root, variable1, "%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]))
        w1.place(x=150,y=100)
        w2 = OptionMenu(self.root, variable2, "%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]))
        w2.place(x=150,y=140)
        def ok():
            tle1 = variable1.get()
            tle2 = variable2.get()
            for i in range(len(title)):
                if title[i]==tle1:
                    val1 = i
            for i in range(len(title)):
                if title[i]==tle2:
                    val2 = i
            print(val1,val2)
            if val1==val2:
                label = tk.Label(self.root, text="Pick the second Article different from the first topic!",fg="red", font=("Arial Bold", 10))
                label.place(x=175,y=170)
            else:
                label = tk.Label(self.root, text="                                                                                  ",fg=
```

```
"red", font=("Arial Bold", 10))
                label.place(x=170,y=170)

                label100 = tk.Label(self.root, text="Paper 1", font=("Verdana", 12))
                label100.place(x=200,y=200)
                label200 = tk.Label(self.root, text="Paper 2", font=("Verdana", 12))
                label200.place(x=660,y=200)
                T1 = tk.Text(self.root,height = 15, width = 40, font=("Helvetica", 12))
                T1.place(x=50,y=250)
                self.quote1 = all_files[val1]
#                self.stringfiles1 = [" ".join([l for l in self.quote1])]
                T1.insert(END, self.quote1)
                T1.config(state=DISABLED)

                T2 = tk.Text(self.root,height = 15, width = 40, font=("Helvetica", 12))
                T2.place(x=500,y=250)
                self.quote2 = all_files[val2]
#                self.stringfiles2 = [" ".join([l for l in self.quote2])]
                T2.insert(END, self.quote2)
                T2.config(state=DISABLED)


                label_1 = tk.Label(self.root, text="Enter text here", font=("Helvetica", 12))
                self.entry_1 = tk.Entry(self.root, font=("Verdana", 10))
                label_1.place(x=80,y=550)
                self.entry_1.place(x=200,y=540,width=500,height = 55)

                logbtn = tk.Button(self.root, text="Predict", command = self._login_btn_clickked,height = 3,width = 10)
                logbtn.place(x=700,y=540)
                logbtn2 = tk.Button(self.root, text="Clear", command = self.buttonPushed,height = 3,width = 10)
                logbtn2.place(x=770,y=540)

        button = Button(self.root, text="OK", command=ok)
        button.place(x=70,y=125)
        self.root.mainloop()
    def buttonPushed(self):
        self.root.destroy()
    def _login_btn_clickked(self):
        self.search1 = self.entry_1.get()
        if len(self.search1)>0:

            self.model1 = model
            self.model2 = model

            self.test_data = word_tokenize(self.search1.lower())
```

```
            v1 = self.model1.infer_vector(self.test_data, steps=20, alpha=0.025)
            v2 = self.model2.infer_vector(self.test_data, steps=20, alpha=0.025)
            v01 = self.model1.infer_vector(self.quote1, steps=20, alpha=0.025)
            v02 = self.model2.infer_vector(self.quote2, steps=20, alpha=0.025)
            distance1 = cosine(v1,v01)
            distance2 = cosine(v2,v02)
            print(distance1,distance2)
            print(self.search1,"@#$@#$")

            if distance1>distance2:
                label300 = tk.Label(self.root, text="                                        ", font=("Helvetica", 14))
                label300.place(x=300,y=600)
                label300 = tk.Label(self.root, text="Closest article is from Paper 1",fg="green", font=("Helvetica", 14))
                label300.place(x=300,y=600)
            else:
                label300 = tk.Label(self.root, text="                                        ", font=("Helvetica", 14))
                label300.place(x=300,y=600)
                label300 = tk.Label(self.root, text="Closest article is from Paper 2",fg="green", font=("Helvetica", 14))
                label300.place(x=300,y=600)

        else:
            pass


LP=LoginPage()
```