

Name – Shobhit Agrawal

Registration Number – 20BDS0162

Stream – B.Tech. CSE with specialization
in Data Science

Digital Assignment – 1

Faculty Name – Dr. Swathi J N

Subject – Information Extraction and
Retreival (TH)

Subject Code – BCD3005

Slot – B1+TB1 – SJT711

Worked with

Siddharth Mandal

20BDS0157

B2+TB2

Write a python program to implement the retrieval models for the query Q = “OpenAI chatbot chatGPT”. Implement KNN approach for text classification.

What is KNN?

K-Nearest Neighbors (KNN) is a simple and intuitive algorithm used for text classification. The KNN algorithm works as follows:

- First, each document in the training dataset is represented as a vector of features, such as word frequency or TF-IDF scores.
- When a new document needs to be classified, the algorithm calculates the distance between the new document and all the documents in the training dataset. This distance can be calculated using Euclidean distance or cosine similarity.
- The KNN algorithm then selects the k-nearest documents to the new document based on the distance metric. These k-nearest documents are referred to as the "nearest neighbors" of the new document.
- Finally, the algorithm assigns the class label of the new document based on the class labels of its nearest neighbors. This can be done using a simple majority vote or by taking the weighted average of the class labels of the neighbors.

For example, if $k=5$ and the 5 nearest neighbors of the new document belong to classes A, A, B, A, and B, then the KNN algorithm would classify the new document as class A, since it has more nearest neighbors in class A.

In text classification, KNN can be used with a variety of feature representations, such as word frequency, TF-IDF scores, or embeddings. KNN is a simple and effective algorithm that can work well for small datasets with few classes, but it may not scale well to large datasets or high-dimensional feature spaces.

Documents:

D1: Since OpenAI released its blockbuster bot ChatGPT in November, users have casually experimented with the tool, with even Insider reporters trying to simulate news stories or message potential dates. To older millennials who grew up with IRC chat rooms — a text instant message system — the personal tone of conversations with the bot can evoke the experience of chatting online. But ChatGPT, the latest in technology known as "large language model tools," doesn't speak with sentience and doesn't "think" the way people do.

D2: Other tech companies like Google and Meta have developed their own large language model tools, which use programs that take in human prompts and devise sophisticated responses. OpenAI, in a revolutionary move, also created a user interface that is letting the general public experiment with it directly. Some recent efforts to use chat bots for real-world services have proved troubling — with odd results. The mental health company Koko came under fire this month after its founder wrote about how the company used GPT-3 in an experiment to reply to users.

D3: The founder of the controversial DoNotPay service, which claims its GPT-3-driven chat bot helps users resolve customer service disputes, also said an AI "lawyer" would advise defendants in actual courtroom traffic cases in real time, though he later walked that back over concerns about its risks. Chat GPT is an AI Chatbot developed by Open AI. The chatbot has a language-based model that the developer fine-tunes for human interaction in a conversational manner. Effectively it's a simulated chatbot primarily designed for customer service; people use it for various other purposes too though. These range from writing essays to drafting business plans, to generating code. But what is it and what can it really do?

D4: Chat GPT is an AI chatbot auto-generative system created by Open AI for online customer care. It is a pre-trained generative chat, which makes use of (NLP) Natural Language Processing. The source of its data is textbooks, websites, and various articles, which it uses to model its own language for responding to human interaction. The main feature of Chat GPT is generating responses like those humans would provide, in a text box. Therefore, it is suitable for chatbots, AI system conversations, and virtual assistants. However, it can also give natural answers to questions in a conversational tone and can generate stories poems and more. Moreover, it can: Write code, Write an article or blog post, Translate, Debug, Write a story/poem, Recommend chords and lyrics

Query:

Q = "OpenAI chatbot chatGPT"

Source Code:

```
import pandas as pd
from nltk.stem import WordNetLemmatizer, SnowballStemmer
from nltk.stem.porter import *
from sklearn.neighbors import NearestNeighbors
import gensim
import string
from sklearn.feature_extraction.text import TfidfVectorizer
import seaborn as sns
```

```

import matplotlib.pyplot as plt

from google.colab import drive
drive.mount('/content/drive')
data=pd.read_csv('/content/drive/MyDrive/Colab Notebooks/dataset.csv')
data.head()
import nltk
nltk.download('stopwords')
nltk.download('omw-1.4')
nltk.download('wordnet')
data['length_doc'] = data['Doc'].str.len()
sns.histplot(data['length_doc'], color="r")
plt.show()

from nltk.corpus import stopwords
stop_words = stopwords.words('english')

stemmer = SnowballStemmer('english')

def lemmatize_stemming(text):
    return stemmer.stem(WordNetLemmatizer().lemmatize(text, pos='v'))

stemmer = SnowballStemmer('english')

def lemmatize_stemming(text):
    return stemmer.stem(WordNetLemmatizer().lemmatize(text, pos='v'))

def preprocess(text):
    result = []
    for token in gensim.utils.simple_preprocess(text):
        regex = re.compile('[ ' + re.escape(string.punctuation) + '0-9\\r\\t\\n']') # removing
punctuations
        text = regex.sub(" ", text.lower()) # lowercasing
        words = text.split(" ") # tokenization
        words = [re.sub("\S*@\S*\s?", "", sent) for sent in words] # special chars
        # words = [re.sub("\s+", ' ', sent) for sent in words]
        words = [re.sub("'", "", sent) for sent in words] # apostrophes
        if token not in stop_words and len(token) > 3: # not stopword && length>3
            result.append(lemmatize_stemming(token)) # lematize and then stem
    return " ".join(result)

preprocessed_docs=[preprocess(doc) for doc in data['Doc']]
print(preprocessed_docs)

query="OpenAI chatbot chatGPT"
query=preprocess(query)
query

vectorizer = TfidfVectorizer()

```

```
vectors = vectorizer.fit_transform(preprocessed_docs) # converting doc to vector (calculating weights)
feature_names = vectorizer.get_feature_names_out() # getting the words
dense = vectors.todense()
denselist = dense.tolist()
df = pd.DataFrame(denselist, columns=feature_names)
query_vector = vectorizer.transform([query]) # convert query to vector

df
```

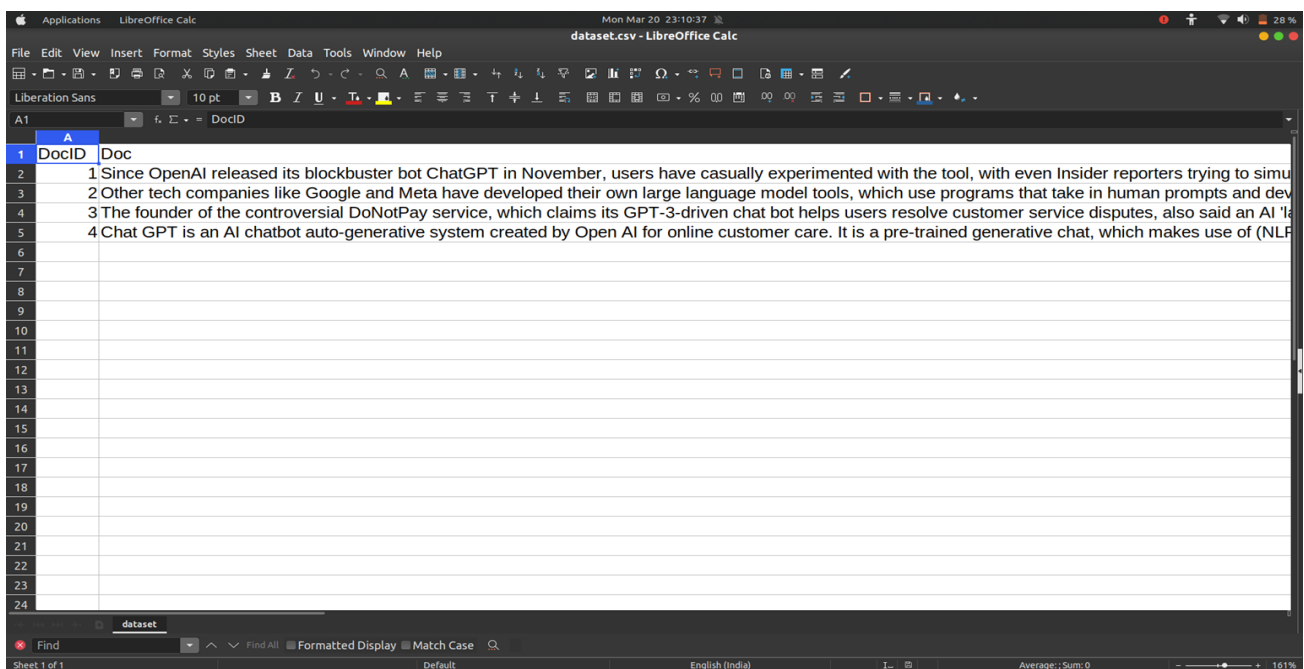
```
k = 2 # Number of nearest neighbors to retrieve (no. of docs to retrieve)
knn_model = NearestNeighbors(n_neighbors=k)
knn_model.fit(vectors)
```

```
distances, indices = knn_model.kneighbors(query_vector) # indices has the index number of the closest docs
```

```
for i in range(k):
    print("Document", indices[0][i]+1, ":", data["Doc"][indices[0][i]])
```

Input:

Document Dataset as a CSV file



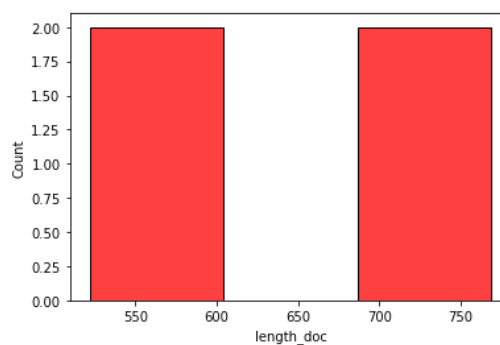
DocID	Doc
1	Since OpenAI released its blockbuster bot ChatGPT in November, users have casually experimented with the tool, with even Insider reporters trying to simulate human-like conversations.
2	Other tech companies like Google and Meta have developed their own large language model tools, which use programs that take in human prompts and develop responses based on the information provided.
3	The founder of the controversial DoNotPay service, which claims its GPT-3-driven chat bot helps users resolve customer service disputes, also said an AI chatbot could be used to help with legal issues.
4	Chat GPT is an AI chatbot auto-generative system created by Open AI for online customer care. It is a pre-trained generative chat, which makes use of (NLP) natural language processing.
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	

Query as input

```
✓ 0s query="OpenAI chatbot chatGPT"
      query=preprocess(query)
      query
      ↗ 'openai chatbot chatgpt'
```

Output:

Document Length Analysis



Preprocessed Docs

```
▼ PREPROCESSING THE DOCS

✓ 2s [9] processed_docs=[preprocess(doc) for doc in data['Doc']]
      print(processed_docs)

      ['sinc openai releas blockbuster chatgpt novemb user casual experi tool even insid report']
```

Preprocessed Query

```
▼ PREPROCESSING THE QUERY

✓ 0s query="OpenAI chatbot chatGPT"
      query=preprocess(query)
      query
      ↗ 'openai chatbot chatgpt'
```

Documents as Vectors (TF-IDF Weights)

TF-IDF SCORES

✓ [12] df

	actual	advis	also	answer	articl	assist	auto	back	base	blockbust	...
0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.147417	...
1	0.000000	0.000000	0.089384	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...
2	0.123989	0.123989	0.079140	0.000000	0.000000	0.000000	0.000000	0.123989	0.123989	0.000000	...
3	0.000000	0.000000	0.072443	0.113496	0.226993	0.113496	0.113496	0.000000	0.000000	0.000000	...

4 rows × 160 columns

Most Relevant Documents

PRINTING THE MOST SIMILAR DOCS

```
for i in range(k):  
    print("Document", indices[0][i]+1, ":", data["Doc"][indices[0][i]])
```

Document 1 : Since OpenAI released its blockbuster bot ChatGPT in November, users have casually
Document 3 : The founder of the controversial DoNotPay service, which claims its GPT-3-driven ch

Working behind inbuilt functions used

TF-IDF

The formula that is used to compute the **tf-idf** for a term t of a document d in a document set is

$$\text{tf-idf}(t, d) = \text{tf}(t, d) * \text{idf}(t)$$

and the **idf** is computed as

$$\text{idf}(t) = \log [n / \text{df}(t)] + 1$$

where n is the total number of documents in the document set and $\text{df}(t)$ is the document frequency of t ; the document frequency is the number of documents in the document set that contain the term t .

Note that the idf formula above differs from the standard textbook notation that defines the idf as

$$\text{idf}(t) = \log [n / (\text{df}(t) + 1)]$$

and **TF**'s variant is **raw frequency** i.e., directly taking the term frequencies without any normalisation.

KNN

Source Code:

```
def __init__(
    self,
    *,
    n_neighbors=5,
    radius=1.0,
    algorithm="auto",
    leaf_size=30,
    metric="minkowski",
    p=2,
    metric_params=None,
    n_jobs=None,
):
```

Metric used for distance computation

Default is "**minkowski**", which results in the **standard Euclidean distance** when $p = 2$.

The Minkowski distance between 1-D arrays u and v , is defined as

$$\|u - v\|_p = \left(\sum |u_i - v_i|^p \right)^{1/p}.$$
$$\left(\sum w_i (|u_i - v_i|^p) \right)^{1/p}.$$

Parameters: u : *(N,)* **array_like**

Input array.

v : *(N,)* **array_like**

Input array.

p : **scalar**

The order of the norm of the difference $\|u - v\|_p$. Note that for $0 < p < 1$, the triangle inequality only holds with an additional multiplicative factor, i.e. it is only a quasi-metric.

w : *(N,)* **array_like, optional**

The weights for each value in u and v . Default is None, which gives each value a weight of 1.0

Reference:

- <https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.distance.minkowski.html#scipy.spatial.distance.minkowski>