Documentation

1) An overview of the function of the code (i.e., what it does and what it can be used for)

The purpose of our code is to provide an alternative to the search functionality of Youtube videos. First, the user will enter a search query that represents the types of videos that they want to search for. Rather than using this query and comparing it to the titles of the Youtube videos, we decided to compare this query to the transcripts of the Youtube videos. This way, users can have an alternative way to search for Youtube videos rather than searching by video title. In order to make this accessible to users, we originally decided to create a chrome extension for this code, so it could be accessible and easy to use for the users. However, after experimenting more with the extension and its implementation, we opted for creating a webpage instead. The use of our webpage is primarily for users seeking to search for more specific information that may struggle to appear in a YouTube title, but may appear in the transcript. This will allow the user to discover a more accurate video pertaining to their search query.

2) Documentation of how the software is implemented with sufficient detail so that others can have a basic understanding of your code for future extension or any further improvement.

The software is run using a file with Flask endpoints called youtube_ranker.py. We print the ranking in order of most relevant to least relevant as organized by the BM25Okapi algorithm on a webpage. Here is the screenshot of the file below:

```python
from flask import Flask, render_template, request
from rank import getRankedURLs
app = Flask(__name__)

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/user', methods=['POST'])
def user():
    query = request.form.get("query")
    titles = getRankedURLs(query)
    return render_template('index.html', titles=titles)
```

There are two endpoints. The first endpoint displays the web page where the user is prompted to type a query, while the second endpoint displays the web page with the ranked YouTube titles.

The implementation of the webpage is located in index.html. A form element is created over the input field to extract the query, which will be fed into youtube_ranker.py to return the ranked titles of youtube videos in our corpus. Then, the ranked titles will be displayed as an ordered list. Here is a screenshot of index.html:

```
1   <h1>Search</h1>
2
3   <form action="{{ url_for('user')}}" method="post">
4       <input type="text" name="query"/>
5   </form>
6
7   <ol>
8       {% for title in titles %}
9           <li>{{title}}</li>
10      {% endfor %}
11  </ol>
12
13
```

To rank the URLs based on the query, a function called getRankedURLs is used, which takes in the argument query. It is located in a file called rank.py. Here is a screenshot of the first 25 lines of code in rank.py

```python
from rank_bm25 import BM25Okapi
from youtube_transcript_api import YouTubeTranscriptApi

def getRankedURLs(query):
    youtube_url_dict = {"5q87K1WaoFI":"Computer Scientist Explains Machine Learning in 5 Levels of Difficulty | WIRED", "z-EtmaFJieY":"Machine Learning & Artificial Intelligence: Crash Course Computer Science #34", "ukzFI9rgwfU":"Machine Learning Basics | What Is Machine Learning? | Introduction To Machine Learning | Simplilearn", "NWONeJKn6kc":"Machine Learning Course for Beginners", "h0e2HAPTGF4":"11. Introduction to Machine Learning"}

    youtube_urls = list(youtube_url_dict.keys())

    corpus = []

    for url in youtube_urls:
        transcript_list = YouTubeTranscriptApi.list_transcripts(url)
        transcript = transcript_list.find_generated_transcript(['en'])
        text = transcript.fetch()

        phrases = ""

        for i in range(len(text)):
            phrases = phrases + ' ' + text[i]['text']

        corpus.append(phrases)

    tokenized_corpus = [doc.split(" ") for doc in corpus]

    bm25 = BM25Okapi(tokenized_corpus)
```

In getRankedURLs, we first have a dictionary with the URL as a key and the title as the values. We obtain the youtube URLs from the dictionary and create a corpus by obtaining the transcripts of the YouTube videos. This is done using a Python package titled youtube_transcript_api.

Next, we use the Okapi BM25 algorithm from the Python package rank_bm25, and return the ranked transcripts of the YouTube transcripts. Based on the ranking of the YouTube transcripts, we translate that to the ranking of the corresponding titles of the YouTube videos. Below is the rest of the implementation of getRankedURLs.

```python
    bm25 = BM25Okapi(tokenized_corpus)

    tokenized_query = query.split(" ")

    doc_scores = bm25.get_top_n(tokenized_query, corpus, n=len(corpus))

    ranks = []

    for i in range(len(doc_scores)):
        url = youtube_urls[corpus.index(doc_scores[i])]
        ranks.append(youtube_url_dict[url])

    return ranks
```

3) Documentation of the usage of the software including either documentation of usages of APIs or detailed instructions on how to install and run a software, whichever is applicable

To set up our code follow the steps below:

First you should clone our code from Github. After this, go to the folder where our code is downloaded. When in here, we recommend using conda to set up a virtual environment. If using conda, use “conda -create -n <env> python=3.5” to create a virtual environment. When this is created, we need to install three packages to this environment. The three packages that need to be installed are flask (the backend framework), rank_bm25 (ranking algorithm), and youtube-transcript-api (to extract the video transcripts). To install these packages simply run the following: “pip install flask”, “pip install youtube-transcript-api”, “pip install rank_bm_25”. After the packages have all been installed, a flask command must be run. If you are running Windows enter “set FLASK_APP=youtube_ranker.py”, otherwise if you are on MacOS enter “export FLASK_APP=youtube_ranker.py”. Finally, you can type in “flask run” to run our code on the webpage. From here you can enter in a query to search and click enter for the results to be displayed.

4) Brief description of contribution of each team member in case of a multi-person team

Snehal - Created Web Crawler to extract Youtube transcripts from the Youtube videos. Helped with all other areas, specifically utilization of flask for the webpage.

Dhyey- Created the ranker to rank the different Youtube videos based off of the user’s query.

Nabil - Created the Chrome extension that allows the user to enter in a query. Chrome extension became confusing/difficult to understand and implement, so the group agreed on continuing with the webpage instead. Created a webpage that allows the user to enter in a query.