Question-Answering System

Shayaan Absar

1 Introduction

The aim of this project is to build an automated Question-Answering system. The model will be passed a collection of documents (*a corpus*) and will perform some linguistic analysis on it (described in section 3). The user will then be able to pass a query into the model and the model will return an answer to their query.

2 Key Concepts

2.1 Tokenization

Tokenization is the process of splitting up text into smaller pieces called tokens. The tokens can be words, sentences or characters.

2.2 Stopwords

Stopwords are words that are found in a sentence but provide no real information. Words such as 'and', 'or', 'the' and 'is' are all examples of stopwords. Generally, stopwords are filtered out when performing any analysis on text.

2.3 Lemmatization

Lemmatization is the process of finding the root form of a word (its *lemma*). For example, 'ran' and 'running' would both become 'run' after lemmatization. This enables these words to be treated the same which yields better results.

The lemma of a word can change depending on how the word is being used. For example, 'building' when used as a verb has a lemma of 'build' but when it is used as a noun, its lemma is 'building'. Therefore, we need to find the *part of speech* of the word to lemmatize it.

2.4 Inverse Document Frequency

Inverse Document Frequency or (idf) is a formula used to find the keywords in a document that is contained within a corpus. The formula determines how unique a word is to the document and the words with the highest values are the keywords.

$$log\left(\frac{\text{Number of Documents}}{\text{Number of Documents containing the word}}+1\right) \quad (1)$$

The more documents that contain the word (the less unique a word is), the closer $\frac{\text{Number of Documents}}{\text{Number of Documents containing the word}}$ is to 1. The logarithm of 1 is 0 meaning that the greater the uniqueness of a word, the higher its idf value.

2.5 Term Frequency-Inverse Document Frequency

Term Frequency-Inverse Document Frequency (tf-idf) is a variant of the idf formula that takes into account the term frequency (number of times a term appears in a document).

$$\text{Term Frequency} \times log\left(\frac{\text{Number of Documents}}{\text{Number of Documents containing the word}}+1\right) \quad (2)$$

3 Model Design

3.1 Text Filtering

Firstly, we need to filter all of the text in our corpus. This involves the following-

1. Loading all of the text into memory.
2. Removing Punctuation.
3. Tokenizing the files into word tokens.
4. Lemmatizing the tokens.
5. Removing stopwords.

3.2 Selecting the Correct Sentence to Answer the Question

After filtering the text, we then calculate the inverse document frequencies for all of the words in the corpus. We can use these values to find the file that is most relevant to the user's query. First, we filter the user's query in the same way that we filtered the text. Then, we calculate the sum of the tf-idf values for all of the words present in the query for each file and then return the file with the highest value (this is the most relevant file).

Then we need to select the correct sentence to answer the question. To do this we need to tokenize the most relevant file into sentences and then calculate the sum of the idf values for all of the words in the query for each sentence and then return the sentence with the highest value.

4 Code with Comments

4.1 Importing libraries and Global Variables

```
from os import listdir, path
from nltk import pos_tag
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.corpus import stopwords, wordnet
from nltk.stem import WordNetLemmatizer
from string import punctuation
from math import log

STOPWORDS = set(stopwords.words('english'))

NUM_DOCUMENTS = len(listdir('corpus'))
files = {}
tokenized_files = {}
idfs = {}
```

4.2 Code for Text Filtering

```
def load_files():
    # Get the names of all files in the corpus
        file_names = listdir('corpus')

        for file in file_names:
                with open(path.join('corpus', file), encoding='utf8') as f:
                        # Place the contents of the file into the files dict
                        files[file] = f.read()
                        # Place the filtered version into tokenized_files dict
                        tokenized_files[file] = filter_text(files[file])

def filter_text(text):
        # Remove punctuation
        for i in punctuation:
                text = text.replace(i, '')
        # Use nltk word tokenizer to tokenize
        text = word_tokenize(text.lower())
        # Use our lemmatize function to lemmatize the text
        text = lemmatize(text)
        # Remove stopwords
        text = [word for word in text if word not in STOPWORDS]

        return text

def lemmatize(text):
        # Use nltk's pos_tag function to find the Part of Speech
        text = pos_tag(text)

        for i, val in enumerate(text):
                word, tag = val
                # The pos tags returned by nltk's pos_tag function
                # are differented to the ones used by the lemmatizer
                # So the tags need to be converted

                match tag[0]:
                        case 'J':
                                tag = wordnet.ADJ
                        case 'V':
                                tag = wordnet.VERB
                        case 'N':
                                tag = wordnet.NOUN
                        case 'R':
                                tag = wordnet.ADV
                        case _:
                                tag = None

                # Use the WordNet lemmatizer to lemmatize the word
                # using the word and the pos tag (if one was found)

                if tag is None:
                        word =  WordNetLemmatizer().lemmatize(word)
                else:
                        word = WordNetLemmatizer().lemmatize(word, tag)

                text[i] = word

        return text
```

4.3 Code for Calculating Inverse Document Frequencies

```
def calculate_idfs():
        # Calculate idf values for all of the words.

        for file in tokenized_files:
                # Create a set of each word in the file (sets remove duplicates)
                words = set(tokenized_files[file])

                # Loop through the set and add 1 to the words count in the idf dict
                for word in words:
                        if word in idfs:
                                idfs[word] += 1
                        else:
                                idfs[word] = 1

        # idfs[i] is currently the number of documents containing the word
        for i in idfs:
                idfs[i] = log(NUM_DOCUMENTS / idfs[i]) + 1
```

4.4 Code for Selecting a Sentence to Answer the Question

```
def top_score(relevancy_scores):
        return max(relevancy_scores, key=lambda x: relevancy_scores[x])

def top_file(query):
        relevancy_scores = {}

        for file in tokenized_files:
                relevancy_scores[file] = 0
                # Find the total tf-idf value for each file
                for word in query:
                        try:
                                relevancy_scores[file] += (idfs[word] *
                                tokenized_files[file].count(word))
                        except KeyError:
                                pass
        # Return file with highest value
        return top_score(relevancy_scores)

def top_sentence(query, top_file):
        relevancy_scores = {}
        #Tokenize the most relevant file into sentences
        sentences = sent_tokenize(files[top_file])

        for sentence in sentences:
                relevancy_scores[sentence] = 0
                words = filter_text(sentence)
                # Find total idf value for each sentence
                for word in query:
                        if word in words:
                                try:
                                        relevancy_scores[sentence] += idfs[word]
                                except KeyError:
                                        pass
        # Return sentence with the highest value
        return top_score(relevancy_scores)
```

4.5 Code for Running the Program

```
def main():
        load_files()
        calculate_idfs()

        while True:
                query = filter_text(input(">>> "))
                print(top_sentence(query, top_file(query)))


main()
```

5 Conclusion

5.1 Testing

I passed a corpus containing 3 text files into the model. The text files were Wikipedia articles for the UK, USA and India. I then asked the model the following questions and received the following answers.

```
>>> How many Olympics took place in the United States?
Eight Olympic Games have taken place in the United States.
>>> Who is the national personification of the United Kingdom?
Britannia is a national personification of the United Kingdom,
originating from Roman Britain.
>>> What is the average life expectancy in India?
The average life expectancy in India is at 70 years to 71.5 years
for women, 68.7 years for men.
>>> What groups is the United Kingdom a member of?
The United Kingdom is a member of the Commonwealth of Nations,
the Council of Europe, the G7, the Group of Ten, the G20, the
United Nations, NATO, AUKUS, the Organisation for Economic
Co-operation and Development (OECD), Interpol, and the World
Trade Organization (WTO).
>>> What sport is most popular in India
Cricket is the most popular sport in India.
>>> What are some widely taught languages in the United States?
The most widely taught foreign languages in the United States,
in terms of enrollment numbers from kindergarten through
university undergraduate education, are Spanish (around 7.2
million students), French (1.5 million), and German (500,000).
>>> What is the India the world's largest producer of?
India is the world's largest producer of milk, with the
largest population of cattle.
```

5.2 Problems with the model

One problem with the model occurs due to the fact that not all of the text in the corpus forms sentences. Some of it is captions, tables or headings. This means the when nltk's sentence tokenizer is used, a heading or image caption may be included in the sentence it is closest to. This means unnecessary information is sometimes included in the answer.

```
>>> What is a traditional thanksgiving dish?
Food
Main article: Cuisine of the United States
A roasted turkey
Roasted turkey is a traditional dish of Thanksgiving dinner.
```

As you can see, the model provides the correct answer but also some unnecessary headings and image captions.

Another problem is that the model fails to recognise synonyms. To improve the model, we could make use of a database that listed synonyms of words and then instead of just looking for words that are in the query we could also look for some common synonyms. Lemmatization helps with this problem slightly but the model could still be improved.