Natural Language Processing (UML602)

Project Report

TEXT SUMMARISER

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Submitted to:

Dr. Ajay Kumar

Submitted by:

Shervil Gupta 101603312

Shivam Mittal 101603316

Table of Contents

NEED ANALYSIS

Availability of ample of data on the web is difficult of access. Hence it has become an important research area of automatic text summarization within the Natural Language processing. Text mining is another important research field that brings meaning to the natural language on the web. Automatic Text Summarization is the Process in which the input is to the computer is the text, whereas the output is the concise extract of the input data. The entire process of automatic text summarization takes four stages. They are tokenization, feature identification, characterization, tagging and summarization. The work can be used in much practical application.

It gets very hectic to read the whole text where most of the sentences are not of much importance and are generally redundant. Text Summarization can also help there as only the valuable information is displayed. Moreover, it is a difficult job to select whether to read a certain article or not. A simple solution to it is to rather read the summary of the article and then decide it if matches the interests of the reader and is it worthy of reading.

ABSTRACT

Text summarization is the process of generating short, fluent, and most importantly accurate summary of a respectively longer text document. The main idea behind automatic text summarization is to be able to find a short subset of the most essential information from the entire set and present it in a human-readable format. As online textual data grows, automatic text summarization methods have potential to be very helpful because more useful information can be read in a short time.

In this new era, where tremendous information is available on the internet, it is most important to provide the improved mechanism to extract the information quickly and most efficiently. It is very difficult for human beings to manually extract the summary of a large documents of text. There are plenty of text material available on the internet. So, there is a problem of searching for relevant documents from the number of documents available and absorbing relevant information from it. In order to solve the above two problems, the automatic text summarization is very much necessary. Text summarization is the process of identifying the most important meaningful information in a document or set of related documents and compressing them into a shorter version preserving its overall meanings.

Text summarization involves reducing a text file into a passage or paragraph that conveys the main meaning of the text. The searching of important information from a large text fileis very difficult job for the users. Thus, to automatically extract the important information or summary of the text file, this summary helps the users to reduce time instead of reading the whole text file and it also provides quick information from the large document. In today's world, to extract information from the World Wide Web is very easy. This extracted information is a huge text repository. With the rapid growth of the World Wide Web (internet), information overload is becoming a problem for an increasing large number of people. Automatic summarization can be an indispensable solution to reduce the information overload problem on the web.

TOOLS AND TECHNOLOGY USED

The tools and technologies used to develop Text Summarizer are:

- Python: It is an interpreted, high level, general purpose language created by Guido Van Rossum. Python is highly extensible, dynamically typed and supports multiple programming paradigms like object-oriented, procedural and functional programming.

- NLTK: NLTK stands for natural language toolkit. This toolkit is one of the most powerful NLP libraries which contains packages to make machines understand human language and reply to it with an appropriate response.

- Flask: Flask is used to create the user interface of the application. It is a micro web framework written in Python. Flask was created by Armin Ronacher of Pocoo.

- Regex: A regex or regular expression is a sequence of characters that define a search pattern. This pattern is usually used by string searching algorithms for input validation or search operations.

METHODOLOGY

We are trying to make a text summarizer which will keep the sentences having major information and neglect the sentences which carry less information.

The frequency of each word is calculated and stored in a word frequency table. Then the sentences are scored where the word frequency of each word of the sentence is added and the outcome is stored as the score of the sentence.

Sentences having score greater than that of average score are kept and all the others are neglected.

It is a basically a five-step process as stated below:

1. **Create the word frequency table:**
   We create a dictionary for the word frequency table from the text. For this, we should only use the words that are not part of the stop Words array.

2. **Tokenize the sentences:**
   Now, we split the text string in a set of sentences. For this we will use the inbuilt method from the NLTK

3. **Score the sentences:**
   to score a sentence by its words, adding the frequency of every non-stop word in a sentence.

4. **Find the best sentences:**
   Here, we are considering the scores of sentences greater than average score.

5. **Generate the summary:**

   Select a sentence for a summarization.

CODE

```
#making the necessary imports
from flask import Flask,render_template,request
import nltk
import re
import heapq


#some helping functions
#------------------------------------------------------------------------------
def clean_the_text(unclean_text):
    unclean_text = re.sub(r'\[[0-9]*\]',' ',unclean_text)
    unclean_text = re.sub(r'\s+',' ',unclean_text)
    clean_text = unclean_text.lower()
    clean_text = re.sub(r'\W',' ',clean_text)
    clean_text = re.sub(r'\d',' ',clean_text)
    clean_text = re.sub(r'\s+',' ',clean_text)
    return clean_text

def generate_word2count(ct,sp):
    word2count = {}
    for word in nltk.word_tokenize(ct):
        if word not in sp:
            if word not in word2count.keys():
                word2count[word] = 1
            else:
                word2count[word] += 1
    for key in word2count.keys():
        word2count[key] = word2count[key]/max(word2count.values())
    return word2count

def score_sentences(s,w):
    sent2score = {}
    for sentence in s:
        for word in nltk.word_tokenize(sentence.lower()):
            if word in w.keys():
                if len(sentence.split(' ')) < 25:
                    if sentence not in sent2score.keys():
                        sent2score[sentence] = w[word]
                    else:
                        sent2score[sentence] += w[word]
    return sent2score
```

```
def do_everything(text):
    #take in the paragraph for summarizing
    sents = nltk.sent_tokenize(text)
    #calling the preprocessing function
    clean_text = clean_the_text(text)
    stop_words = nltk.corpus.stopwords.words('english')
    #creating word2count table
    word2count = {}
    word2count = generate_word2count(clean_text,stop_words)
    #give score to the sentences
    sent2score = {}
    sent2score = score_sentences(sents,word2count)
    #summarize the text according to scored sentences and print the result
    #first approach: using average score
    average_score = sum(list(sent2score.values()))/len(list(sent2score.values()))
    best_sentences2 = []
    for i in list(sent2score.keys()):
        if sent2score[i] >= average_score:
            best_sentences2.append(i)
    return best_sentences2
#--------------------------------------------------------------------------
app = Flask(__name__)
@app.route("/")
def show_main_page():
    return render_template('main.html')
@app.route("/action",methods=["POST","GET"])
def summarize():
    x = request.form["t1"]
    print(x)
    lis = []
    lis = do_everything(x)
    summary = lis[0]
    for i in range(1,len(lis)):
        summary += lis[i]
    return render_template('main.html',summary=summary)

if __name__ == '__main__':
   app.run()

#second approach: using heapq
#best_sentences = heapq.nlargest(5, sent2score, key=sent2score.get)
#print('----------------------------------------------------------')
#for sentence in best_sentences2:
    #print(sentence)
```

RESULT

With the ever-growing text data, text summarization seems to have the potential for reducing the reading time by showing summaries of the text documents that capture the key points in the original documents. Applying text summarization on each article can potentially improve customer experience and employees' productivity. The tools we made explored the possibility of implementing text summarization on large dataset. While some datasets had decent summaries generated by our model, there are several ways the model can be improved further. The model we built for abstractive summarization did a good job on generating human readable sentences from given inputs. However, it did not always generate summaries capturing all the important information in the input documents. Further research has to be done in on this, to get the solution to the above encountered problem.