document

October 12, 2019

Page Summarizer (Task 7)

Abstract

This project is a task from the HNG Machine learning Internship 2019. The model built in this notebook gives an abstractive summary of a page/article. Basically the model is trained to go through the page, get the major information from it and then give a summary of the page. Explanations of how the model works and how it is trained is presented in the notebook.

Libraries Used

We import the necessary libraries needed to manipulate the data, for numerical computation and for building our model.

- RegEx - RegEx is imported as re. A RegEx, or Regular Expression, is a sequence of characters that forms a search pattern. RegEx can be used to check if a string contains the specified search pattern. It is imported as re.
- Gensim - Gensim is a Python library for topic modelling, document indexing and similarity retrieval with large corpora. It is a useful library for NLP.
- Numpy - Numpy is used for scientific computations. It is imported as np.
- Sklearn - Scikit-learn is a library in Python that provides many unsupervised and supervised learning algorithms. In this project we make use of the cosine similarity to measure similarity.

In [2]:
```
#import necessary libraries
import re
import gensim
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
import networkx as nx
```

```
C:\ProgramData\Anaconda3\lib\site-packages\gensim\utils.py:1197: UserWarning: detected Windows
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

Reading Data

Our data is contained in "mayowa.txt" file and we load it and save it as file object

In [3]:
```
#reading file
file = open("mayowa.txt","r")
data=file.readlines()
file.close()
```

0.0.1 Preprocessing of Data

A function is defined below to process the text. The function first converts the text to lowerstring (Capital to small letter, then it checks if any of [([].*?[])] is in the new string and removes any of them found. The function then removes whitespaces and also removes numbers from the string. The processed strings which now consist of just lower cased words is now split into a list (tokens). The function then takes the list of words joins them with a whitespace and them removes them.

```
In [4]: #define preprocessing steps
        #lower case
        #remove everything inside []
        #remove 's
        #fetch only ascii characters

        def preprocessor(text):
            newString = text.lower()
            newString = re.sub("[\(\[].*?[\)\]]", "", newString)
            newString = re.sub("'s","",newString)
            newString = re.sub("[^'0-9.a-zA-Z]", " ", newString)
            tokens=newString.split()
            return (" ".join(tokens)).strip()
```

The function above is called below and for each text in the data is passed/processed through it and saved in the list (text). Each text saved in the list is then seperated with a (.). Each of the sentences is then checked for whitespaces and if any is found it is stripped.

```
In [5]: #call above function
        text=[]
        for i in data:
            text.append(preprocessor(i))

        all_sentences=[]
        for i in text:
            sentences=i.split(".")
            for i in sentences:
                if(i!=''):
                    all_sentences.append(i.strip())
```

The sentences are then tokenized to be used for training in the model

```
In [6]: # tokenizing the sentences for training word2vec
        tokenized_text = []
        for i in all_sentences:
            tokenized_text.append(i.split())
```

The model used for training is the word2vec model from the Gensim library and various parameters are speculated

```
In [7]: #define word2vec model
        model_w2v = gensim.models.Word2Vec(
```

```
                    tokenized_text,
                    size=200, # desired no. of features/independent variables
                    window=5, # context window size
                    min_count=2,
                    sg = 0, # 1 for cbow model
                    hs = 0,
                    negative = 10, # for negative sampling
                    workers= 2, # no.of cores
                    seed = 34)
```

Model is then trained on the tokenized text

```
In [8]: #train word2vec
        model_w2v.train(tokenized_text, total_examples= len(tokenized_text), epochs=model_w2v.
```

```
Out[8]: (19002, 35240)
```

A function is defined below to obtain sentence embedding. It works by first creating a numpy array of zeros in the size variable specified and then check if word is present in vocabulary. If word is not present continue if count is not zero and vec is not equal to count.

```
In [9]: #define function to obtain sentence embedding
        def word_vector(tokens, size):
            vec = np.zeros(size).reshape((1, size))
            count = 0.
            for word in tokens:
                try:
                    vec += model_w2v[word].reshape((1, size))
                    count += 1.
                except KeyError: # handling the case where the token is not in vocabulary

                    continue
            if count != 0:
                vec /= count
            return vec
```

```
In [10]: #call above function
         wordvec_arrays = np.zeros((len(tokenized_text), 200))
         for i in range(len(tokenized_text)):
             wordvec_arrays[i,:] = word_vector(tokenized_text[i], 200)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:7: DeprecationWarning: Call to
  import sys
```

Similarity between both word vectors arrays is then computed

```
In [11]: # similarity matrix
         sim_mat = np.zeros([len(wordvec_arrays), len(wordvec_arrays)])
```

```
In [12]: #compute similarity score
         for i in range(len(wordvec_arrays)):
           for j in range(len(wordvec_arrays)):
             if i != j:
               sim_mat[i][j] = cosine_similarity(wordvec_arrays[i].reshape(1,200), wordvec_arra
```

A graph of the similarity is also generated

```
In [13]: #Generate a graph
         nx_graph = nx.from_numpy_array(sim_mat)

In [14]: #compute pagerank scores
         scores = nx.pagerank(nx_graph)

In [16]: #sort the scores
         sorted_x = sorted(scores.items(), key=lambda kv: kv[1],reverse=True)

         sent_list=[]
         for i in sorted_x:
             sent_list.append(i[0])
```

A sample Summary is presented below to test model

```
In [17]: #extract top 10 sentences
         num=10
         summary=''
         for i in range(num):
             summary=summary+all_sentences[sent_list[i]]+'. '
         print(summary)
```

```
the highlights for him included hitting a hundred with a swollen jaw and helping india avoid th
```

0.1 Task 8

For task 8 a function is defined which that takes in the url of an article and returns the text in the url.

0.1.1 Libraries used

Beautiful Soup library which is a very useful library for webscraping is used for this task.

```
In [ ]: import requests
        from bs4 import BeautifulSoup

        def read_content():
            """

            :returns: text-> blah
```

```
        :rtype: string

        """
        url = "https://en.wikipedia.org/wiki/Machine_learning"
        response = requests.get(url)
        html = response.content
        soup = BeautifulSoup(html)
        text = soup.text
        return text


    print(read_content())
```

0.2 Task 9

Function that takes in title of article obtained from url, passes it through the page summarizer model and saves summary as a .txt file

0.2.1 Libraries Used

Beautiful Soup - For scrapping data from web **NLTK** - Natural language toolkit helps build NLP models. **Pandas** - For manipulating data **Numpy** - For scientific computation

```
In [ ]: # importing libraries

        import numpy as np
        import pandas as pd
        from bs4 import BeautifulSoup
        import nltk
        import urllib.request
        from nltk.corpus import stopwords
        from nltk.tokenize import word_tokenize, sent_tokenize
        from string import punctuation
        from heapq import nlargest
        from collections import defaultdict
```

- Downloading stopwords and pukt package from Natural language toolkit

```
In [ ]: #Downloading the stopwords package
        nltk.download('stopwords')
        nltk.download('punkt')
```

- Data used is obtained from Lucid blog post, 'On Time with Chibuike Osita'

```
In [ ]: # Scrapping Data from Lucid blog On Time with Chibuike Osita

        url="https://lucid.blog/hngi6/post/qa-with-chibuike-osita-49e"
        requested_url = urllib.request.urlopen(url).read().decode('utf8','ignore')
```

- Using beautiful soup to read data from url

```
In [ ]: # read the data from the url

        soup= BeautifulSoup(requested_url, 'html.parser')
```

- removing all text extracted from url with p tag

```
In [ ]: # find all text that has p tag
        text_p = soup.find_all('p')
        print(text_p)
```

- making each text present lower cased and also tokenizing the words

```
In [ ]: for i in range(0,len(text_p)):
            text += text_p[i].text
        text = text.lower()
        # tokenize the text
        tokens =[t for t in text.split()]
        print(tokens)
```

- Removing irrelevant words, numbers or punctuations

```
In [ ]: clean_token =tokens[:]
        #define irrelevant words that include stop words , punctuations and numbers
        stopword = set(stopwords.words('english') + list(punctuation))
        for token in tokens:
            if token in stopword:
                clean_token.remove(token)

        print(clean_token)
```

- Obtaining the frequency distribution of the words

```
In [ ]: freq = nltk.FreqDist(clean_token)
        top_words=[]
        top_words=freq.most_common(100)
        print(top_words)
```

```
In [ ]: sentences = sent_tokenize(text)
        print(sentences)
```

- Creating a ranking for each sentence

```
In [ ]: #Iterating through all the sentences from the web to create a ranking for each sentenc

        ranking = defaultdict(int)
        for i, sent in enumerate(sentences):
            for word in word_tokenize(sent.lower()):
                if word in freq:
                    ranking[i]+=freq[word]
            top_sentences = nlargest(10, ranking, ranking.get)
        print(top_sentences)
```

```
In [ ]: #printing one of the top 2 sentences
        print(sentences[27])
```

- Sorting through the sentences

```
In [ ]: #Sorting all sentences
        sorted_sentences = [sentences[j] for j in sorted(top_sentences)]
        print(sorted_sentences)
```

```
In [ ]: with open('sorted_sentences.txt', 'w') as f:
            for i in sorted_sentences:
                f.write(i+"\n")
```

0.3 Task 10

API to handle text summarizer in task 9. It receives url as input and gives the summary of the article in the url as output

```
In [ ]: import pickle as p
        # import traceback
        from flask import Flask, request, jsonify
        import json
        app = Flask(__name__)

        #@app.before_request
        #def exe():
        #    summarizer = 'modelfile.pkl'
        #    model = p.load(open(summarizer, 'rb'))


        @app.route('/api/summarize', methods=['POST', 'GET'])
        def get_url():
            """

            :returns: jsonified-> blah
            :rtype: string

            """
            if request.method == 'POST':
                url = request.json['theUrl']
                #print(content)
                jsonified = jsonify(url), 200
                return jsonified


        if __name__ == '__main__':
            app.run(debug=True, host='127.0.0.1', port=5000)
```