Plagiarism Checker

According to Wikipedia, Plagiarism means stealing or copying the words of another as one's own or using someone's product without crediting the source or to present a new idea derived from the existing source. So all in all, Plagiarism is an act of fraud and could lead to legal crisis if not dealt with properly.

There isn't any clear cut rules as to how much percent of plagiarism is accepted in a document. By convention, journals accept a text similarity of less than 15% (macthing text shouldn't be a continuous block of copied material), office documents accept around 5%.

There are several tools available online to check plagiarism content for free but due to policies of their own, they store people's data and sell it to third parties. Inorder to protect the data, a simple plagiarism checker is developed and deployed as a WebApp using Flask.

Currently, the program can compare the given text(query) with articles present on Web.

Methodology

- NLTK library for tokenizing and removing stopwords from the query.
- Webscrapping for finding articles with similar content as the query.
- Cosine Similarity to check the similarity content between query and the listed article.

In [1]:

```
import nltk
import requests
from bs4 import BeautifulSoup as bs
import warnings
import re

warnings.filterwarnings('ignore')
```

Let's have the list of all English stopwords

In [2]:

```
stop_words = set(nltk.corpus.stopwords.words('english'))
```

Let's create a function 'searchEngine' to check the content similar to the query! The function performs the following steps:

- Tokenize the query into list of words, remove stopwords from the list and join the words to form sentence(s)
- Use requests to query using Google search
- Use Beautiful Soup[1] to scrap all the urls with similar content as query.
- return top 2 urls with content similar to query

In [3]:

```
def searchEngine(query):
    """
    This function takes in the query and
    returns the top 2 urls with content similar to query
    """

    # Tokenize the query into list of words, remove stopwords from the list and
 join the words to form sentence(s)
    words = nltk.word_tokenize(query)
    sentences = (' '.join([word for word in words if word not in stop_words]))

    # Use requests to query using Google search
    url = 'https://google.com/search?q=' + sentences
    urls = []
    page = requests.get(url).text

    # Use Beautiful Soup to scrap all the urls with similar content as query
    soup = bs(page, 'lxml')
    for link in soup.find_all('a'):
        url = link.get('href')
        if url.startswith('/url?q'):
            s_link = url.split('=')[1]
            urls.append(s_link)

    return (urls[:2]) # top 2 urls
```

Let's create a function 'extractFromWeb' to scrap the data from the top 2 urls and store it in the database.txt file.

In [4]:

```
def extractFromWeb(url):
    """ This function takes in the url
    and scraps it's text content, stores it in database.txt file"""

    page = requests.get(url).text
    soup = bs(page, 'html.parser')
    with open('database.txt', 'a') as f:
        for i in soup.find_all('p'):
            f.write(str(i.text))
```

Now that the similar content from web is available in the .txt file, let's check the percentage of similarity between the query and .txt file using 'Cosine Similarity' [2]. Cosine similarity is a metric used to measure how similar the documents are irrespective of their size. Mathematically, it measures the cosine of the angle between two vectors projected in a multi-dimensional space. In this context, the two vectors are arrays containing the word counts of query and .txt file.

The cosine similarity is advantageous because even if similar documents are far apart by the Euclidean distance because of the size (like, the word might appear 50 times in one document and 10 times in the query) they could still have a smaller angle between them. Smaller the angle, higher the similarity.

It's possible to use the Scikit Learn's package to compute cosine similarity, but for the sake of better understanding, implementation is done from scratch. Below is a function defined as 'cosineSimilarity' which does the following:

- Get a list of all the unique words from .txt file and query
- Find the frequency of each words in the query and .txt file with respect to the unique words from previous step
- Compute the cosine similarity using the formula[3]

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

where, A = frequency counts of words in query and B = frequency counts of words in .txt file. A_i and B_i are the corresponding word vector components

In [5]:

```
def cosineSimilarity(query, database_file):
    """ Computes the cosine similarity between the query and the given document
    returns the similarity in percent (%) """

    # Get a list of all the unique words from .txt file and query
    unique_words = []
    query_l = re.sub("[^\w]", " ", query.lower()).split()
    v1 = [unique_words.append(word) for word in query_l if word not in unique_wo
rds]

    with open(database_file, 'r') as f:
        database_l = re.sub("[^\w]", " ", f.read().lower()).split()
    v2 = [unique_words.append(word) for word in database_l if word not in unique
_words]

    # Frequency counters
    query_f = []
    database_f = []
    for word in unique_words:
        qc, dc = 0,0

        for w in query_l:
            if w == word:
                qc += 1
        query_f.append(qc)

        for w in database_l:
            if w == word:
                dc += 1
        database_f.append(dc)

    # Compute cosine similarity using above formula
    dot_prod, query_mag, database_mag = 0,0,0
    for i in range(len(query_f)):
        dot_prod += query_f[i] * database_f[i]
        query_mag += query_f[i]**2

    query_mag = (query_mag)**0.5

    for i in range(len(database_f)):
        database_mag += database_f[i]**2
    database_mag = (database_mag)**0.5

    return ((float)(dot_prod / (database_mag * query_mag))*100) # match percenta
ge
```


Let's define a function 'queryWeb' which calls all the above function and returns the match percentage.

In [6]:

```
def queryWeb(text):

    """Takes in the query and returns the match percentage"""

    sites = searchEngine(text)
    matching_sites = []
    for i in sites:
        matching_sites.append(str(i).split('&')[0]) # Top 2 urls identified

    for i in range(len(matching_sites)):
        extractFromWeb(matching_sites[i]) # database is now ready with contents
 from top 2 urls

    matches = cosineSimilarity(text, 'database.txt') # compute similarity

    return (matching_sites,matches)
```

Now that all the functions are defined, let's evaluate the plagiarism checker using 2 examples.

Case 1:

A paragraph taken from textbook (https://jakevdp.github.io/PythonDataScienceHandbook/05.11-k-means.html). On running the checker, this should show >50% result!

In [14]:

```
para = """Those two assumptions are the basis of the k-means model. We will soon
dive into exactly how the algorithm reaches this solution, but for now let's tak
e a look at a simple dataset and see the k-means result. First, let's generate a
two-dimensional dataset containing four distinct blobs. To emphasize that this i
s an unsupervised algorithm, we will leave the labels out of the visualizatio
n"""
```

In [15]:

```
sites, out = queryWeb(para)
print('Match percentage: {}%'.format(round(out,2)))
print('Source urls:')
print(sites[0] +'\n'+ sites[1])
```

```
Match percentage: 73.92%
Source urls:
https://jakevdp.github.io/PythonDataScienceHandbook/05.11-k-means.ht
ml
https://colab.research.google.com/github/jakevdp/PythonDataScienceHa
ndbook/blob/master/notebooks/05.11-K-Means.ipynb
```

Great! This works.

Case 2:

Now let's check another example.

In [16]:

```
para2 = """Most of the applications like internet banking or e-commerce or medic
al records have most sensitive
data which needs to be kept secret. Such data is prone to be hacked by any indiv
idual or any organization"""
#snipet from my college paper
```

In [17]:

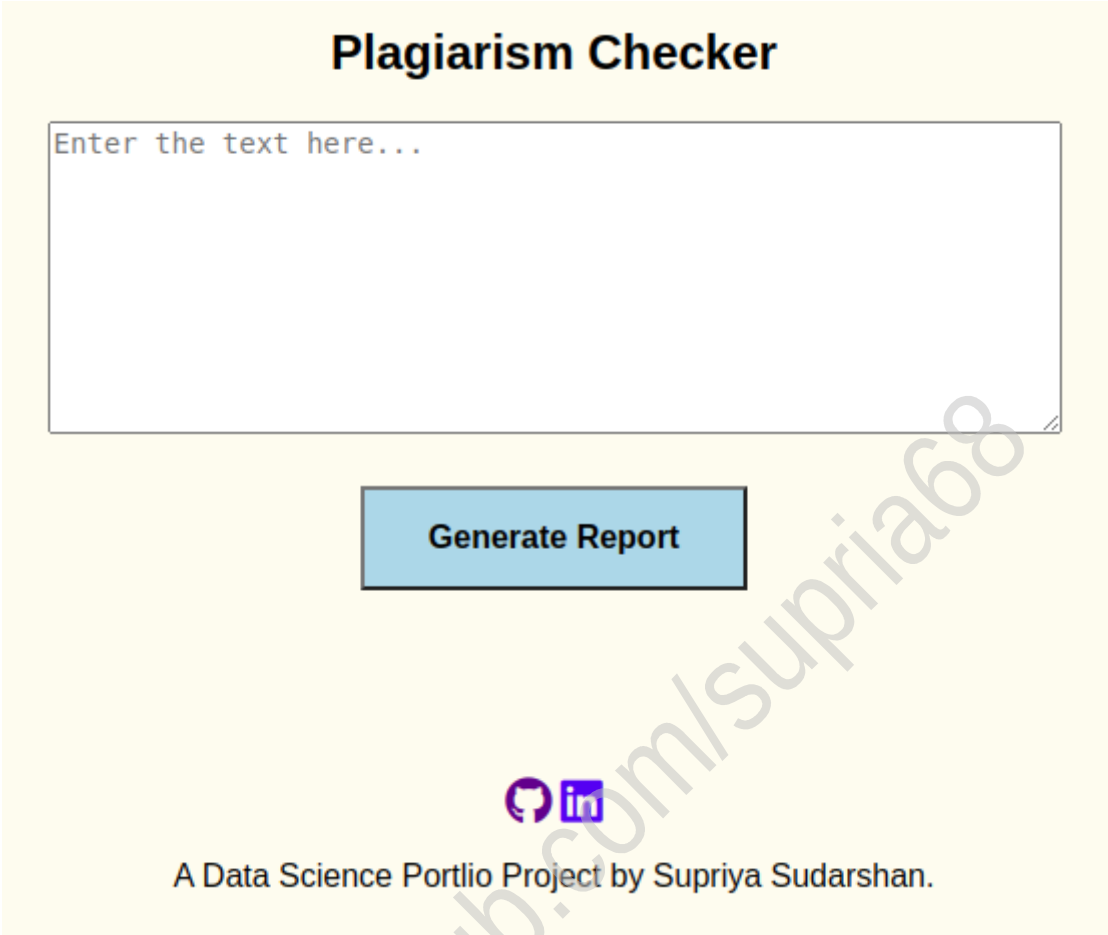
```
# delete the contents of .txt file before performing another check
with open('database.txt', 'w') as f:
    f.write('')

site, out2 = queryWeb(para2)
print('Match percentage: {}%'.format(round(out2,2)))
print('Source urls:')
print(site[0] +'\n'+ site[1])
```

```
Match percentage: 7.16%
Source urls:
https://www.researchgate.net/publication/322345990_Big_healthcare_da
ta_preserving_security_and_privacy
https://www.researchgate.net/publication/319938035_Big_data_security
_and_privacy_in_healthcare_A_Review
```

Deploying checker as a WebApp using Flask

Below is a snapshot of the deployed WebApp.



Steps to Execute

- Make sure you have all the libraries and packages as mentioned in the requirements.txt
- Run `python3 app.py`
- Open the browser and go to URL : http://127.0.0.1:5000/ (http://127.0.0.1:5000/)
- Enter the text and click on the 'Generate Report' button to get the match percentage and source urls.

References

[1] WebScrapping using Beautiful Soup (https://www.youtube.com/watch?v=ng2o98k983k&ab_channel=CoreySchafer)
[2] Cosine Similarity (https://www.machinelearningplus.com/nlp/cosine-similarity/)
[3] Formula for Cosine Similarity (https://en.wikipedia.org/wiki/Cosine_similarity)