Natural Language Processing

NatGeo Magnum Opus

Team – 2

Roll Number	Name
17BCE0729	Rutvik R Patel
17BCE0740	Shobit G
17BCE0984	Sparsh Srivastava (Leader)
17BCE2037	Swati Singhvi
17BCE2184	Fiza Rasool
17BCE2200	Swathi Sriram
17BCE2318	Rajan Sahani
18BCB0038	Siddharth Garg
18BCB0043	Christeen T Jose
18BCE0265	Ishaan Ohri

Submitted to,

Prof. Sharmila Banu K

Problem Statement:

Assume you are a part of the NLP Tech team that works for a Publishing House. There is a shortlisted applicant (with her writing samples) for the Editor-in-chief position. How can you help the publishing house with the decision on hiring this applicant?

1) Pipeline

1. Scraping article from Internet
2. Clean the web page
3. Pre-process the article (Stop words removal, tokenize, lemmatize, etc.)
4. Bag-of-words for doc corpus
5. TF of doc corpus
6. IDF vector for terms
7. TF-IDF for doc corpus
8. Normalized TF-IDF for doc corpus
9. Similarity Prediction (Euclidean Prediction, Cosine Similarity)
10. Set threshold for determining candidate

2) Overview of Steps

a) Scrapping Article from Internet and Removing HTML Tags

We used web scrapping to fetch article from the internet which can be subsequently used for final similarity prediction. After fetching data we remove unnecessary tags and meta data and preserve article content for best results. Every removal process is encapsulated in it's individual function as uses beautiful soup for parsing.

Snippet:

```
def remove_html_tags(text):
    clean = re.compile('<.*?>')
    return re.sub(clean, '', text)
def remove_newline(text):
    clean = re.compile('\n')
    return re.sub(clean, '', text)
def remove_refs(text):
    clean = re.compile('\[.*\]')
    return re.sub(clean, '', text)

def extractContentByTag(soup, TAG):
    contents = []
    if TAG == 'p':
        contents = ''
        for tag in soup.find('div', {'class:', 'storyWrap'}).findAll(TAG):
            contents += (remove_refs(remove_newline(remove_html_tags(tag.getText()
))))
        contents = contents[ : contents.rfind('\xa0To\xa0subscribe\xa0to\xa0Nation
al Geographic Traveller India\xa0and\xa0National\xa0Geographic')]

    else:
        for parentTag in soup.findAll('div', {'class:', 'cDescription'}):
            for tag in parentTag.findAll(TAG):
```

```
                if 'href' in tag.attrs.keys():
                    if not tag.attrs['href'].startswith('http', 0): #Filter useles
s URLs
                        continue
                    contents.append(tag.attrs['href'])

    return(contents)

def extractURLs(seedURL):
    req = requests.get(seedURL)
    soup = BeautifulSoup(req.content, 'html5lib')
    URLs = extractContentByTag(soup, 'a')
    return(URLs)

def extractArticles(URLs):
    corpus = {}
    reqs = [requests.get(URL) for URL in URLs]
    soups = [BeautifulSoup(req.content, 'html5lib') for req in reqs]
    corpus = {URLs[i] : extractContentByTag(soups[i], 'p') for i in range(len(URLs
))}
    return(corpus)
```

We fetch all the articles available on Nat Geo website by the writer Lakshmi Sankaran to show comparison scores between multiple articles for more accurate scores and comparisons. In total we get scrape 27 articles.

Snippet:

```
URLs = extractURLs('http://www.natgeotraveller.in/author/lakshmi-sankaran/')
corpus = extractArticles(URLs)
df = pd.DataFrame([(URL, len(corpus[URL])) for URL in corpus], index = ['Article '
 + str(i + 1) for i in range(len(corpus.keys()))], columns = ['Article Link', 'Art
icle Length'])
```

b) Preprocessing the articles

Now we process the 27 articles fetched by removing stopword, lemmatizing using WordNet and stemming using LancasterStemmer, each again encapsulated in their little functions which help us preprocess data into corpus

Snippet:

```
def lemmatizeUsingWordNet(words):
    lemmatizer = WordNetLemmatizer()
    for word in words:
        final.append(lemmatizer.lemmatize(word))
```

```
def stemUsingLancasterStemmer(article):
    stemmer = LancasterStemmer()
    return [" ".join([stemmer.stem(j) for j in i.split()]) for i in article]

def getStopWords():
    StopWords = set(nltk.corpus.stopwords.words('english'))
    StopWords.update(set(punctuation))
    StopWords.update(set(['a','they','the','his','so','and','were','from','that','
of','in','only','with','to']))
    return(StopWords)
```

c) Bag of Words approach for Corpus Document

Having processed the scraped articles we now work towards changing the data into corpus and subsequently in to document. We choose bag of words approach for assembling the document into vectors. We then use this document to calculate our similarity (after few more steps)

Snippet:

```
processedCorpus = {}
processedCorpusKeys = []
bag = bagOfWords(corpus, processedCorpus, processedCorpusKeys)
df = pd.DataFrame(bag, index = [URL for URL in URLs])
```

d) Making a function to calculate TF Matrix of Doc Corpus

After having the corpus as Bags of Words, we make a function that calculates TF Matrix (Term-Frequency) out of the corpus. This step tells us about how frequent a word occurs across all 27 articles better helping us estimate the similarity at a later stage.

Snippet:

```
def getTFMatrix(bag, processedCorpus, processedCorpusKeys):
    totals = [bag['total_terms'][i] for i in range(len(processedCorpusKeys))]
    TFMatrix = {}
    TFMatrix = {term : [bag[term][i] / totals[i] for i in range(len(processedCorpu
sKeys))] for term in bag.keys()}
    del TFMatrix['total_terms']
    return(TFMatrix)
```

e) Making a function to calculate IDF Vector

Similar to TF, we also make a function to calculate IDF Vector(Inverse Document Frequency) which tells us how rare a word is across all the documents. More rareness means that the document is less plagiarised and more unique.

Snippet:

```
def getIDFVector(bag, processedCorpus, processedCorpusKeys):
    IDF = {}
    terms = []
    for article in processedCorpusKeys:
        terms.extend(processedCorpus[article])
    terms = set(terms)
    for term in terms:
        appears = [0 for _ in range(len(processedCorpusKeys))]
        for i in range(len(processedCorpusKeys)):
            if term in processedCorpus[processedCorpusKeys[i]]:
                appears[i] = 1
        IDF[term] = 0 if sum(appears) == 0 else math.log((1 + len(processedCorpusK
eys)) / sum(appears))
    return(IDF)
```

f) Making function to calculate TF-IDF of the corpus

Similar to above 2 functions, we also make a function to calculate the TF-IDF of the entire corpus using the functions that calculate TF vector and IDF matrix. This function helps us to create a matrix which tells us how important a word is in the give articles.

Snippet:

```
def getTF_IDFMatrix(TF, IDV, processedCorpusKeys):
    TF_IDF = {}
    for term in TF:
        TF_IDF[term] = [TF[term][i] * IDV[term] for i in range(len(processedCorpus
Keys))]
    return(TF_IDF)
```

g) Calculating TF-IDF of the corpus

Using the above defined helper functions we calculate the final TF-IDF matrix of the Corpus Documents made from Articles. We store the matrix as a pandas dataframe for final similarity deduction.

Snippet:

```
TF = getTFMatrix(bag, processedCorpus, processedCorpusKeys)
df = pd.DataFrame(TF, index = URLs)
IDV = getIDFVector(bag, processedCorpus, processedCorpusKeys)
df = pd.DataFrame(IDV, index = ['IDF values'])
TF_IDF = getTF_IDFMatrix(TF, IDV, processedCorpusKeys)
df = pd.DataFrame(TF_IDF, index = URLs)
```

h) Normalizing the values in Matrix

We normalize the values in the TF-IDF Matrix in order to bring the values to a comparable scale and eliminate extremities and other biases.

Snippet:

```
def normalize(TF_IDF, processedCorpusKeys):
    norm = {}
    denos = [0 for _ in range(len(processedCorpusKeys))]
    for i in range(len(processedCorpusKeys)):
        denos[i] += sum([TF_IDF[term][i] ** 2 for term in TF_IDF])
    for i in range(len(denos)):
        denos[i] = denos[i] ** 0.5
    for term in TF_IDF:
        norm[term] = [TF_IDF[term][i] / denos[i] for i in range(len(processedCorpu
sKeys))]
    return(norm)
norm = normalize(TF_IDF, processedCorpusKeys)
df = pd.DataFrame(norm, index = ['Article ' + str(i + 1) for i in range(len(proces
sedCorpusKeys))])
```

i) Finding the Final Similarities

We used 2 similarity algorithms to calculate similarity between the documents. All similarity combinations allow us see the similarities between any of 2 of 27 articles (all combinations totalling in at 351 entries). Let's explore result of both the algorithms.

a) Cosine Similarity

Snippet:

```
def getDocumentCosineSimilarityResult(norm, processedCorpusKeys):
    similarityRes = {}
    for i in range(0, len(processedCorpusKeys)):
        for j in range(i + 1, len(processedCorpusKeys)):
            similarity = 0
```

```
            for term in norm:
                similarity += norm[term][i] * norm[term][j]
            res = 'Cosine similarity of URL ' + str(i + 1) + ' with URL ' + str(j
+ 1) + ' is: '
            similarityRes[res] = similarity
    return(similarityRes)

result = getDocumentCosineSimilarityResult(norm, processedCorpusKeys)
df = pd.DataFrame([result.keys(), result.values()], index = ['Article Pair', 'Cosi
ne Similarity'], columns = ['Comparision ' + str(i + 1) for i in range(len(result.
values()))]).transpose()
df.sort_values('Cosine Similarity', axis = 0, ascending = False, inplace = True, k
ind = 'quicksort')
print('Key = higher the value of cosine similarity (angle - dot product), the more
 similar an article pair is')
```

Output:

```
Key = higher the value of cosine similarity (angle - dot product), the more similar an article pair is
```

	Article Pair	Cosine Similarity
Comparision 349	Cosine similarity of URL 25 with URL 26 is:	0.0967496
Comparision 107	Cosine similarity of URL 5 with URL 14 is:	0.092669
Comparision 1	Cosine similarity of URL 1 with URL 2 is:	0.0880026
Comparision 350	Cosine similarity of URL 25 with URL 27 is:	0.0813811
Comparision 325	Cosine similarity of URL 20 with URL 22 is:	0.069155
...	...	...
Comparision 136	Cosine similarity of URL 6 with URL 22 is:	0.00670025
Comparision 129	Cosine similarity of URL 6 with URL 15 is:	0.00579299
Comparision 241	Cosine similarity of URL 12 with URL 22 is:	0.00494994
Comparision 158	Cosine similarity of URL 7 with URL 24 is:	0.00485182
Comparision 292	Cosine similarity of URL 16 with URL 23 is:	0.00442628

351 rows × 2 columns

b) Euclidean Similarity

Snippet:

```
def getDocumentEuclideanDistanceResult(norm, processedCorpusKeys):
    similarityRes = {}
    for i in range(0, len(processedCorpusKeys)):
        for j in range(i + 1, len(processedCorpusKeys)):
            similarity = 0
            for term in norm:
                similarity += math.pow(norm[term][i] - norm[term][j], 2)
            similarity = math.pow(similarity, 0.5)
            res = 'Euclidean distance of URL ' + str(i + 1) + ' with URL ' + str(j
 + 1) + ' is: '
```

```
            similarityRes[res] = similarity
    return(similarityRes)

result = getDocumentEuclideanDistanceResult(norm, processedCorpusKeys)
df = pd.DataFrame([result.keys(), result.values()], index = ['Document Pair', 'Euc
lidean Distance'], columns = ['Comparision ' + str(i + 1) for i in range(len(resul
t.values()))]).transpose()
df.sort_values('Euclidean Distance', axis = 0, ascending = True, inplace = True, k
ind = 'quicksort')
print('Key = lesser the value of Euclidean distance (geometric distance in n-
dimentional Euclidean space), the more similar an article pair is')
```

Output:

```
Key = lesser the value of Euclidean distance (geometric distance in n-dimentional Euclidean space), the more similar an article pair is
```

	Document Pair	Euclidean Distance
Comparision 349	Euclidean distance of URL 25 with URL 26 is:	1.34406
Comparision 107	Euclidean distance of URL 5 with URL 14 is:	1.34709
Comparision 1	Euclidean distance of URL 1 with URL 2 is:	1.35055
Comparision 350	Euclidean distance of URL 25 with URL 27 is:	1.35545
Comparision 325	Euclidean distance of URL 20 with URL 22 is:	1.36444
...	...	...
Comparision 136	Euclidean distance of URL 6 with URL 22 is:	1.40947
Comparision 129	Euclidean distance of URL 6 with URL 15 is:	1.41011
Comparision 241	Euclidean distance of URL 12 with URL 22 is:	1.41071
Comparision 158	Euclidean distance of URL 7 with URL 24 is:	1.41078
Comparision 292	Euclidean distance of URL 16 with URL 23 is:	1.41108

351 rows × 2 columns

Comparing both the tables, we clearly see that the similarity calculated between all the article combination have same result. Which verifies and solidifies that the data is consistent and results are reproducible.
Moreover, the documents are hardly similar in the usage of words and very unique with almost no plagiarism. It can be concluded that the articles are fresh every single time and hence, the author, Lakshmi Sankaran should be hired.

3) Individual Contributions

a) Web-Scraping => Swati Singhvi, Fiza Rasool
b) Cleaning and pre-processing => Shobit G, Rajan Sahani
c) Conversion of raw articles to corpus document => Swathi Sriram, Siddharth Garg
d) Result deduction using Cosine Similarity => Christeen T Jose, Ishaan Ohri
e) Result deduction using Euclidean Similarity => Rutvik R Patel
f) Documentation and Result/Code Compilation => Sparsh Srivastava

Link to Colab for combined code :
https://colab.research.google.com/drive/11NG3pONnfFy_W4x6_SzphB1W52z5u9B1?usp=sharing