Akash Gupta - Technical note

A demonstration of some custom-made processes and functions to enable sentiment analysis on political opinion articles.

Text in itself as data is quite wild and unstructured. In order to start making sense out of it and conduct meaningful analyses, we must convert textual data into machine-readable formats that enable statistical computations. Some of the key methods used in these set of programs and functions are :-

1. Cleaning out html tags from text using **regular expressions**.
2. Tokenizing text into words (unigrams). Tokenizing text into sentences.
3. Parts of speech tagging - a process that involves tagging the grammarly aspects of words.
4. Converting tokenized text into vectors and matrices - using concepts like **word frequency, TfIdf weighting**.
5. Computing similarity of words and documents using measures of **cosine similarity**.
6. Extracting **Nouns** and important keywords from text.

First we start off by loading various python libraries that will enable us to manipulate textual data.

```
import pandas as pd
import nltk
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
import numpy as np
from gensim.models import Word2Vec
from sklearn.decomposition import PCA
from matplotlib import pyplot
from sklearn.metrics.pairwise import cosine_similarity
import math
import spacy
import re
```

Now, data is read into Python and converted into a Dataframe, with the rows containing many documents (in this case opinion articles).

```
df = pd.read_csv('/Users/Akashgupta/Desktop/Rnlp/rpd/forvader.csv')
df = df[['title','text']]
```

Below is a function that implements **regular expression syntax** to clean up the text.

```
import re
for i in range(len(df['text'])):
    df['text'][i] = re.sub(r'[^\x00-\x7f]',r'',df['text'][i])
```

This is a function written especially to extract **proper nouns** after tokenizing the text into sentences. The idea is to break up a large piece of text into sentences and then extract their Nouns. We are extracting Nouns because they essentially serve as the **subjects** regarding whom, sentiment/emotion is judged or arrived at.

```
big_nouns = []
for i in range(18):
    text = nltk.word_tokenize(df['text'][i])
    tagged_news = nltk.pos_tag(text)
    big_nouns.extend(nltk.FreqDist(word for (word,tag) in tagged_news if tag.startswith("NNP")
                                   and word not in ['Telegram','News','App','@','Click','Express',
                                          'download','Opinion','indianexpress','channel',
                                          'headlines'] and len(word)>3))
```

Another loop that tokenizes text and extracts **verbs** from the text.

```
text = nltk.word_tokenize(df['text'][18])
tagged_news1 = nltk.pos_tag(text)
TOIO = list(nltk.FreqDist((word,tag) for (word,tag) in tagged_news1 if tag.startswith("VB")
                         and word not in ['Telegram','News','App','@','Click','Express',
                                         'download','Opinion','indianexpress','channel',
                                         'headlines'] and len(word)>3))
```

Below is a function that stores all the extracted **nouns** into dictionary format. This is done because it is quite smooth to convert dictionary object to **dataframes**. We are converting them to dataframes because Dataframes provide some good functionality in terms of data manipulation.

```
export_dict = {}
export_dict = {"word":big_nouns, 'index':[num for num in range(len(big_nouns))]}
```

```
nouns_post = pd.DataFrame(export_dict)
nouns_post.to_csv("/Users/Akashgupta/Desktop/Rnlp/rpd/nounsofTOI.csv")
```

Another loop that extracts certain words that might be of particular interest to the human.

```
for i in range(36,54):
    for sen in nltk.sent_tokenize(df['text'][i]):
        if 'NREGA' in nltk.word_tokenize(sen):
            noun_list.append(sen)
        elif 'MG-NREGA' in nltk.word_tokenize(sen):
            noun_list.append(sen)
        elif 'Mahatma' in nltk.word_tokenize(sen):
            noun_list.append(sen)
        elif 'MGNREGA' in nltk.word_tokenize(sen):
            noun_list.append(sen)
        elif 'MGNREGS' in nltk.word_tokenize(sen):
            noun_list.append(sen)
        elif 'MNREGA' in nltk.word_tokenize(sen):
            noun_list.append(sen)
```

Another couple of functions and variables that convert noun dictionaries to dataframes.

```
noun_dat = pd.DataFrame({'newht_nouns':noun_list})
noun_dat.to_csv("/Users/Akashgupta/Desktop/Rnlp/rpd/newht_nouns.csv")
```

```
df3 = pd.read_csv("/Users/Akashgupta/Desktop/Rnlp/rpd/forvader.csv")
```

```
df3['title'] = df3['title'] + df3['num'].astype(str)
```

```
df1 = df3[['title','text']]
```

```
import spacy
```

This mini-program essentially **vectorizes** the textual data into a matrix that contains **TfIdf** weights as features.

```
from sklearn.feature_extraction.text import TfidfVectorizer

count_vect = TfidfVectorizer(stop_words = 'english')
count_vect = TfidfVectorizer()
sparse_matrix = count_vect.fit_transform(df1['text'])

doc_term_matrix = sparse_matrix.todense()
df2 = pd.DataFrame(doc_term_matrix,
                  columns = count_vect.get_feature_names(),
                  index = df1['title'])
```

Here a **cosine similarity** function is called and the output can be seen.

```
from sklearn.metrics.pairwise import cosine_similarity
print(cosine_similarity(df2,df2))
```

```
[[1.         0.40487255 0.39944517 ... 0.40239531 0.42733832 0.41218007]
 [0.40487255 1.         0.47716387 ... 0.5174505  0.49382875 0.53039056]
 [0.39944517 0.47716387 1.         ... 0.55904388 0.47559622 0.50436836]
 ...
 [0.40239531 0.5174505  0.55904388 ... 1.         0.47272137 0.51314154]
 [0.42733832 0.49382875 0.47559622 ... 0.47272137 1.         0.52156551]
 [0.41218007 0.53039056 0.50436836 ... 0.51314154 0.52156551 1.        ]]
```

This is a custom made **cosine similarity** function.

```
import numpy as np
from numpy import dot
from numpy.linalg import norm

def cosine_simi(v1, v2):
    if norm(v1)>0 and norm(v2)>0:
        return (dot(v1,v2)/(norm(v1)*norm(v2)))
    else:
        return 0.0
```

A small demonstration of this function as it finds the similarity between the words **modi** and **pain**. It turns out that there is very little similarity between these words - which further suggests that negative words like **pain** are not used too often with referrences to the Prime Minister.

```
cosine_simi(df2['modi'],df2['pain'])
```

```
0.08849279154434557
```

Here is a list of tokens/terms that we want to compare with the word **modi** and rank them based on their **cosine similarity** value.

```
tokens = list(['relief','support','strong','hard','helped','unprepared','pain','promised'])
```

```
def spacy_closest(token_list, vec_to_check, n = 20):
    return sorted(token_list
                 ,key = lambda x: cosine_simi(vec_to_check, df2[x])
                 ,reverse = True)[:n]
```

Turns out that some of the top correlated words with **modi** are - **strong, hard, helped, support** - which further suggest that many positive opinions are associated with the Prime Minister. However there are also negative words like **unprepared** in his context, they are used much less in comparison to the positive referrences.

```
spacy_closest(tokens, df2['modi'])
```

```
['strong',
 'hard',
 'helped',
 'support',
 'unprepared',
 'relief',
 'promised',
 'pain']
```

```
tokens = list(['reforms','benefits','relief','fear','plight','lack','vulnerable',
              'panic','distress','weak','bad','helped','blame','failed',
              'failure','positive','progress','pandemic'])
```

```
ntokens = list([w.lower() for doc in df1['text'] for w in nltk.word_tokenize(doc)
               if w.isalpha() and w not in nltk.corpus.stopwords.words('english')
               and len(w) > 4 and w not in ['migrant','workers']])
```

```
spacy_closest(tokens, df2['modi'])
```

```
['reforms',
 'helped',
 'pandemic',
 'relief',
 'fear',
 'lack',
 'panic',
 'benefits',
 'progress',
 'blame',
 'vulnerable',
 'bad',
 'weak',
 'plight',
 'distress',
 'failure',
 'positive',
 'failed']
```

Further analysis has been done in RStudio. I have basically used Python to carry out basic tasks like - extracting Nouns and sentences and cleaning up texts. I found that RStudio had some more smooth functionality and visualization options. So the main analysis has been done there.

This note is primarily meant to demonstrate some of the functions and processes that I attempted to develop.