Ben Roth - NLP Jupyter Notebook

April 27, 2020

1 Assignment 13

Find your favorite news source and grab the article text.

1. Show the most common words in the article.
2. Show the most common words under a part of speech. (i.e. NOUN: {'Bob':12, 'Alice':4,})
3. Find a subject/object relationship through the dependency parser in any sentence.
4. Show the most common Entities and their types.
5. Find Entites and their dependency (hint: entity.root.head)
6. Find the most similar words in the article

```
In [1]: #import packages
        import spacy
        import pandas as pd
        import numpy as np
        from collections import Counter
        from sklearn.metrics.pairwise import cosine_similarity
```

```
In [2]: #download model
        !python3 -m spacy download en
```

```
Requirement already satisfied: en_core_web_sm==2.0.0 from https://github.com/explosion/spacy-mo

    Linking successful
    /Users/Broth/anaconda3/lib/python3.7/site-packages/en_core_web_sm -->
    /Users/Broth/anaconda3/lib/python3.7/site-packages/spacy/data/en

    You can now load the model via spacy.load('en')
```

For my assignment, I will using an article from Crooked Media entitled "Democrats Can Change History or Doom Us To Repeat It" found (here)[https://crooked.com/articles/democrats-trump-plague/]

```
In [3]: #read in text file
        with open('../data/democrats_can_change_history_or_doom_us_to_repeat_it.txt', 'r') as f
```

```
            text = file.read().replace('\n', '').replace('\\', '').replace("{\rtf1\ansi\ansicp

        print(text[:200])
```

```
In 2018, Republicans lost a statewide vote for the Wisconsin Assembly to Democrats by a wide ma
```

```
In [4]: processor = spacy.load('en')

        processed_text = processor(text)
        processed_text[:200]
```

```
Out[4]: In 2018, Republicans lost a statewide vote for the Wisconsin Assembly to Democrats by a
```

1.0.1 1. Show the most common words in the article.

```
In [5]: tokens = [word.text for word in processed_text if word.is_stop != True and word.is_punc


        # five most common tokens
        word_count = Counter(tokens)
        most_common = word_count.most_common(5)

        print('The most common words and corresponding counts are:')
        print(most_common)
```

```
The most common words and corresponding counts are:
[('Republicans', 16), ('Trump', 16), ('nt', 15), ('Democrats', 14), ('s', 14)]
```

1.0.2 2. Show the most common words under a part of speech. (i.e. NOUN: {'Bob':12, 'Alice':4,})

```
In [6]: parts = [word.pos_ for word in processed_text if word.is_stop != True and word.is_punc
```

```
In [7]: #create dataframe of words and parts of speech
        df = pd.DataFrame({'Word': tokens,
             'PartOfSpeech': parts
            })

        #get count of each word per p.o.s.
        dfgrp = df.groupby(['Word', 'PartOfSpeech']).size().reset_index(name='Count')

        #get most common words under a part of speech
        print("The two most common words per part of speech are:")
        print(dfgrp.sort_values(['PartOfSpeech','Count'], ascending = False).groupby('PartOfSp
```

```
The two most common words per part of speech are:
            Word PartOfSpeech  Count
```

```
310          hoc             X      1
618         vote          VERB      8
102        allow          VERB      4
0                        SPACE      1
516       second         PUNCT      1
71         Trump         PROPN     16
55   Republicans         PROPN     14
68          They          PRON      9
36            It          PRON      4
643            s          PART     10
1            180           NUM      1
2             20           NUM      1
246     election          NOUN     13
195  coronavirus          NOUN      7
66           The           DET      8
65          That           DET      3
14           But         CCONJ      5
11           And         CCONJ      3
409           nt           ADV     15
375       matter           ADV      3
34            If           ADP      4
64          That           ADP      2
22    Democratic           ADJ      9
53    Republican           ADJ      3
```

1.0.3 3. Find a subject/object relationship through the dependency parser in any sentence.

```
In [8]: sentences = [sentence for sentence in processed_text.sents]

        first_sentence = sentences[0]

        first_sentence

Out[8]: In 2018, Republicans lost a statewide vote for the Wisconsin Assembly to Democrats by a

In [9]: dependencies = {}

        for word in first_sentence:
            #subject would be
            if "subj" in word.dep_:
                dependencies[word] = word.dep_
            #iobj for indirect object
            elif "obj" in word.dep_:
                dependencies[word] = word.dep_
            else:
                continue
        print('In the first sentence of the article:')
        for key, value in dependencies.items():
            print('    - The word', key, 'is a', value)
```

```
In the first sentence of the article:
   - The word Republicans is a nsubj
   - The word vote is a dobj
   - The word Assembly is a pobj
   - The word Democrats is a pobj
   - The word margin is a pobj
   - The word majority is a dobj
   - The word seats is a pobj
   - The word themselves is a dobj
   - The word accountability is a pobj
   - The word voters is a pobj
```

1.0.4 4. Show the most common Entities and their types.

```
In [10]: entities = [entity for entity in processed_text.ents]
         labels = [entity.label_ for entity in processed_text.ents]

         df1 = pd.DataFrame({'Word': tokens})

         df2 = pd.DataFrame({'Entity': entities,
              'Type': labels
             })

         df2['Entity'] = df2['Entity'].astype(str).str.replace(')', '').str.replace("(", "")

         df = pd.merge(df1, df2, left_on = 'Word', right_on = 'Entity', how = 'left')

         dfgrp = df.groupby(['Word', 'Type']).size().reset_index(name='Count')

         print('The ten most common entities and their types are:')
         print(dfgrp.sort_values('Count', ascending = False).head(10))
```

```
The ten most common entities and their types are:
           Word    Type  Count
22  Republicans    NORP    256
13    Democrats    NORP    196
25        Trump    NORP     96
29    Wisconsin     GPE     90
12   Democratic    NORP     90
26        Trump     ORG     48
27        Trump  PERSON     32
10     Congress     ORG     25
16          GOP     ORG      9
```

```
24       Senate     ORG      9
```

1.0.5 5. Find Entites and their dependency (hint: entity.root.head)

```
In [14]: ent_dep = [entity.root.head.text for entity in entities]


         df = pd.DataFrame({
             'Entity':entities,
             'Dependency':ent_dep
         })

         print(df.head(5))
```

```
                       Entity Dependency
0                      (2018)         In
1               (Republicans)       lost
2  (the, Wisconsin, Assembly)        for
3                 (Democrats)         to
4                (This, week)       used
```

1.0.6 6. Find the most similar words in the article

I tried using the spaCy similarity function, but I kept getting the following error:

```
ValueError: [E010] Word vectors set to length 0. This may be because you don't have a model ins
```

Because of this, I used sklearn's cosine similarity function to find similarities. Unfortuntaley, this can only find similarities for exact words, not necessarily in meaning.

```
In [12]: sentences = [[token.orth_ for token in sentence] for sentence in processed_text.sents

         cos_sims = pd.DataFrame(columns = ['Sentence1', 'Sentence2', 'Similarity'])
         i = 0

         for x in range(len(sentences)):
             for y in range(len(sentences)):

                 sent1_list = sentences[x]
                 sent1_count = dict(Counter(sent1_list))
                 sent1_count['Sentence'] = 'sent1'

                 sent2_list = sentences[y]
                 #count sentences and create id key
                 sent2_count = dict(Counter(sent2_list))
                 sent2_count['Sentence'] = 'sent2'
```

```
            #create dfs, and get vectorized sentence values
            df1 = pd.DataFrame(sent1_count, index = ['sent1'])
            df2 = pd.DataFrame(sent2_count, index = ['sent2'])

            df = pd.concat([df1, df2], axis=0, ignore_index=True, sort = False)
            df = df.fillna(0)

            vals1 = list(df[df['Sentence'] == 'sent1'].drop(columns = ['Sentence'], axis
            vals2 = list(df[df['Sentence'] == 'sent2'].drop(columns = ['Sentence'], axis


            #find similarities and add to output df
            vals1 = list(df[df['Sentence'] == 'sent1'].drop(columns = ['Sentence'], axis
            vals2 = list(df[df['Sentence'] == 'sent2'].drop(columns = ['Sentence'], axis

            similarity = cosine_similarity(vals1, vals2)

            cos_sims.loc[i] = [x, y, round(similarity[0][0], 3)]

            i += 1
```

```
In [13]: cos_filt = cos_sims[cos_sims['Similarity'] < 1.0].sort_values(['Similarity'], ascendi

         cos_head = cos_filt.head(1)

         fin_sen1 = int(cos_head['Sentence1'].values[0])
         fin_sen2 = int(cos_head['Sentence2'].values[0])

         print('The two most similar sentences are: ')
         print('"', ' '.join(sentences[fin_sen1]), '",')
         print('and:')
         print('"', " ".join(sentences[fin_sen2]),'"')
```

```
The two most similar sentences are:
" First , protect the election from the pandemic , and then win it by a wide - enough margin t
and:
" Against the backdrop of the plague election in Wisconsin , and Republican efforts to hobble
```