Speech Analysis

Andrey Verbovskiy

Background:

In this research I would like to analyze one of the most impactful and powerful speeches in order to identify the most versatile method for speech writing. I plan on doing a few continuous researches on this topic, this is the first one of them.

Personal interest:

In future I certainly will have to present some speeches in public, if this and following speech researches provide some logical conclusions, it will be a great help to me in speechwriting.

Some of the speeches are quite important for history, as one of my hobbies is history studies, this is a great opportunity for me to better understand the feelings/emotions of people of those ages.

Goal:

In the first of my speech analysis researches I plan on analyzing the affection patterns changes during the 10 famous speeches. The key point of this first research is to estimate the positivity/negativity/neutrality of text, calculate the score of those values and use the scores to estimate the delta3 value for a clear representation.

Hypothesizes:

Hypothesize 1: There is a certain affection pattern that the speech writers deliberately or unconsciously use in order to make a powerful speech.

Hypothesize 2: Speech's affection pattern is dependable on the time period of its writing and the topic, so this research will give just an overall view and requires future specification of data selections.

Method:

The first stage of this research was the data preparation. Opening the txt files:

```
file = 'william_wilberforce.txt'
william_wilberforce = open(file,'r',encoding="utf8").read()
```

```
file = 'lincoln.txt'
lincoln = open(file,'r',encoding="utf8").read()
```

```
file = 'subhas_chandra_bose.txt'
subhas = open(file,'r',encoding="utf8").read()
```

```
file = 'patrick_henry.txt'
henry = open(file,'r',encoding="utf8").read()
```

```
file = 'nelson_mandela.txt'
nelson_mandela = open(file,'r',encoding="utf8").read()
```

```
file = 'martin_luther_king.txt'
luther = open(file,'r',encoding="utf8").read()
```

```
file = 'gandhi.txt'
gandhi = open(file,'r',encoding="utf8").read()
```

```
file = 'demosthenes.txt'
demosthenes = open(file,'r',encoding="utf8").read()
```

```
file = 'churchill.txt'
churchill = open(file,'r',encoding="utf8").read()
```

```
file = 'emmeline_pankhurst.txt'
pankhurst = open(file,'r',encoding="utf8").read()
```

Then, I created a function that prepares and tokenizes (splits text into separate words-tokens) the opened text for future analysis. Then, it lemmatizes and outputs the final tokens.

```
def preparation(text):

    for c in ['.',',',':',';','-','&', '/', 'my']:
        text = text.replace(c,'')

    tokens = text.split()
    tokens=[token.lower() for token in tokens if token.isalpha()]
    ctokens = tokens
    sw = stopwords.words('english')

    for token in tokens:
        if token in sw:
            ctokens.remove(token)

    lemmatizer = WordNetLemmatizer()

    for i in range(len(ctokens)):
        ctokens[i] = lemmatizer.lemmatize(ctokens[i])
    return ctokens
```

After the function we are left with key words instead of plain text, the main reasoning behind this process is that we do not need meaningless junk words (for example prepositions etc) that hurt the accuracy of our analysis.

Next stage is opening the positive and negative words files, luckily I was able to find those files already prepared in required format and there is no need in using our preparation function. Those positive and negative words are essential for our analysis, because we will check if key word from the analyzed speeches belong to any of those two lists.

```
#opened file with the list of positive words for futher analysis
file_positive = 'p.txt'
pos = open(file_positive,'r').read().split()
```

```
#opened file with the list of negative words for futher analysis
file_negative = 'n.txt'
neg = open(file_negative,'r').read().split()
```

Main method:

Now, we need to create a loop that identifies if the word is positive or negative, if it does not belong to any of these categories it is counted as a neutral word. This method allows me to fully categorize the text content and illustrates the affection proportions of the text. However, one limitation of this method is that it will most likely make neutral part the biggest one in terms of proportion, yet it seems logical even when considering our daily speech where we mostly use neutral words to deliver our message and then emphasize our feelings/emotions with positive/negative words.

```
def affection (ctoken):
    countpositive = countnegative = countneutral = counttotal = 0
    for token in ctoken:
        counttotal = counttotal + 1
        if token in pos:
            countpositive = countpositive + 1
        elif token in neg:
            countnegative = countnegative + 1
        else:
            countneutral = countneutral + 1
    print("Positive=%.3f Negative=%.3f Neutral=%.3f" % (countpositive/counttotal, countnegative/counttotal, countneutral/count
total))

    print("Positiveness=%.3f Affection=%.3f" % (math.log(countpositive/countnegative), 1-countneutral/counttotal))
```

To represent the affection change more smoothly we need delta3 score, it is way better in terms of representation than just plotting the plane affection score as it not only represents the affection of a current word but also a shift of affection score from its closest 5 neighbors. For that we firstly need to split tokens in groups and calculate the group's overall score. This function does its job perfectly:

```
def scores(ctokens,number_of_words_in_one_group):
    score = []
    countpositive1 = countnegative1 = countneutral1 = counttotal1 = counttotal2 = 0
    count = 0
    for token in ctokens:
        counttotal1 = counttotal1 + 1
        if(counttotal1 % number_of_words_in_one_group == 0):
            if token in pos:
                countpositive1 = countpositive1 + 1
                count = countpositive1 - countnegative1
                score.append(count)
                count = 0
                countpositive1 = 0
                countnegative1 = 0
            elif token in neg:
                countnegative1 = countnegative1 + 1
                count = countpositive1 - countnegative1
                score.append(count)
                count = 0
                countpositive1 = 0
                countnegative1 = 0
            else:
                countneutral1 = countneutral1 + 1
                count = countpositive1 - countnegative1
                score.append(count)
                count = 0
                countpositive1 = 0
                countnegative1 = 0

        else:
            if token in pos:
                countpositive1 = countpositive1 + 1
            elif token in neg:
                countnegative1 = countnegative1 + 1
            else:
                countneutral1 = countneutral1 + 1
    return score
```

This function requires two parameters: the prepared tokens and the number of words in one group. As data speeches have different word lengths, they also have different number of tokens, by listing the numbers of tokens I realized that they all have dividers in range of 20-30, which means that the number of tokens in each group can be divided by some value in this small area and so we can get approximately the same number of groups for each text. So, the number of words in group is simply number of tokens/divider(in range of 20-30).

In next section we are calculating the delta3 score. The formula of delta3 is:

Delta3(x[i]) = (x[i]+ x[i+1] + x[i+2] + x[i-1] + x[i-2])/5

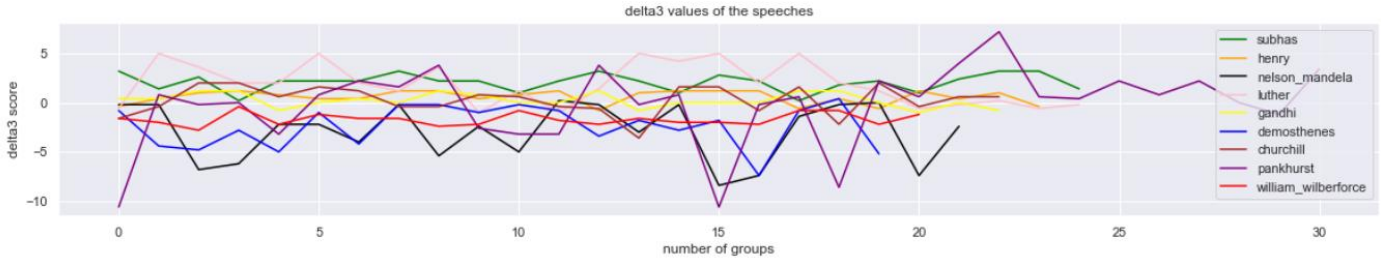
```
# Counting the delta values in the radious of 5
def delta3(scores):
    delta3 = []

    for i in scores:
        cv = 0
        cv = scores[i] + scores[i-1] + scores[i+1] + scores[i-2] + scores[i+2]
        cv = cv/5
        delta3.append(cv)
    return delta3
```

The last part of this research is the analysis of the plotted affection delta3 scores.

Firstly, we set the picture size to be wider to make plots more understandable.

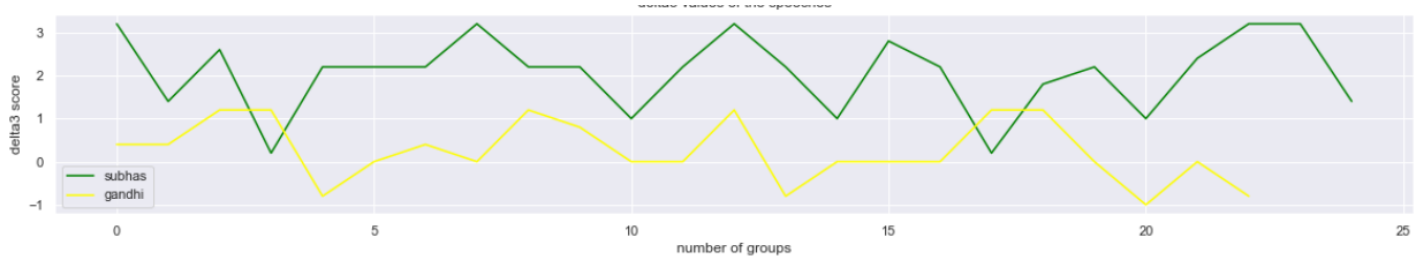
Secondly, we plot all the graphs on one picture to see if there are any patterns:



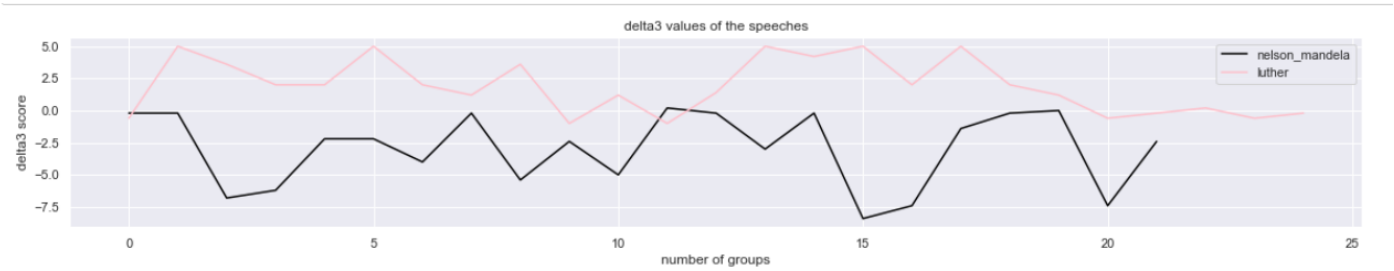
Observations:

We can see some trends on the graph, but mostly all the approaches are quite unique and not versatile. In the beginning authors try to keep it neutral with small deviations, the main action takes place near the middle groups where authors make it either strongly negative or positive. Closer to the end all authors tend to end their speeches in neutral manner. Judging from the graph there are unfortunately not that many patterns to identify.

However, if we limit the number of speeches to ones that are connected by timeline or topic or both, things get much more interesting. For example if we take Mahatma Gandhi and Subhas Chandra Bose speeches:



As you can see, sometimes the graph patterns are almost identical, the difference is that Gandhi's speech is much more neutral than that of Subhas. Both have almost same shifts, especially the last ones that end almost identically. These two speeches are addressed to India's independence movement and the publishing dates are not far away from each other. So, does it mean that limitation of topic and publishing time indeed helps in tracking the affection patterns? This can really help in improving speechwriting techniques. Let's take another example of Nelson Mandela and Martin Luther King:



Again, there is a resemblance between these two speeches in terms of trending, especially in the first half. The main difference if that First speech is quite negative and hardly-ever becomes neutral, while the second one is a complete opposite in this perspective.

Conclusion and future improvements:

Make the speech selection more topic and time period oriented. The main reason is that the selected speeches have are too far in terms of timeline and topics. One of the possible theories is that the manner of speech changes over the time, or does it not? In future analysis I should test this theory. Next time I should narrow my list of speeches to the one consisting of certain topic and short time difference. In order to test if it has any impact.

Use the acquired knowledge in building a neural network which can write a speech based on affection patterns changes.