# Advanced Natural Language Processing Assignment 3

**Group :**
Anusha Ramamurthy
Anup Bharadwaj
Suhas Jagadish
Raghuveer Krishnamurthy Kanchibail
Supreeth Surya Prakash

## Introduction

We used the following two Texts:

1. Pride and Prejudice
2. The three Musketeers

We choose the above two novels because they were both set in Europe. They were published in the 1813, all though The Three musketeers  is set in the 1625 - 1628.

We discussed multiple approaches and decided to go with two.

## Implementation

We installed the Standford Core NLP package on Windows Operating Systems.  We used the following annotators

1. lemma
2. Name Entities Recognizer (ner)
3. Coreferences (using the Deterministic Coreference Annotator dcoref)
4. deppparse

Approaches: We split our approaches to analyse and look for patterns in speaker conversations in two broad ways. One Semi-automatically and the other completely automatic using programming.

## Approach One:

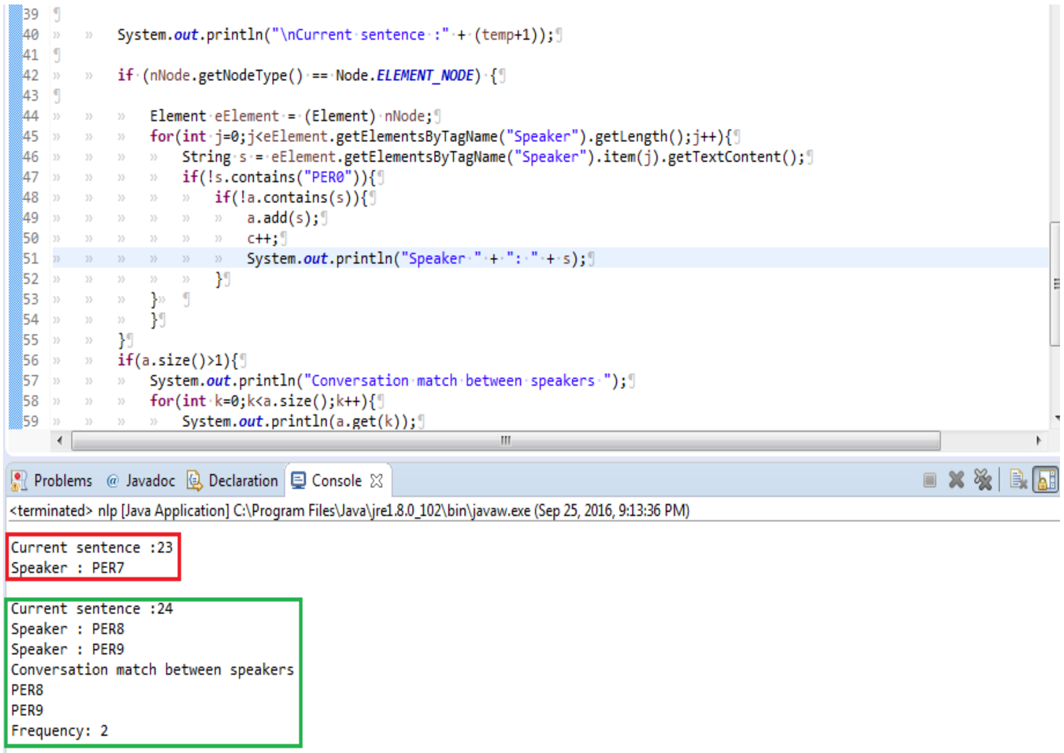
We tried a fully automatic approach to detect the frequencies with which, different characters are speaking to each other.

Below are the steps followed –

- Downloaded the novels “Pride & Prejudice” (for development) and “The three musketeers” (for testing).
- Downloaded the Stanford CoreNLP library and generated the xml file using the below command,

  **java -cp "*" -Xmx2g edu.stanford.nlp.pipeline.StanfordCoreNLP – annotators tokenize,ssplit,pos,lemma,ner,parse,dcoref,depparse -file test.txt**

- The generated xml file contains a tag named **<Speaker>** which gives us the speaker names for each tokens.
- The xml file is structured in a way that the document is divided into number of sentences, each sentence is divided into number of tokens and the **<Speaker>** tag is generated for each of the token.
- The attached java program (nlp.java) reads this xml document as a text using the in-built java DOM parses.
- The code checks for total number of speakers involved in each sentence and ignores the speaker “PER0” since it is the narrator.
- If there are more than 1 speakers involved in a sentence (ignoring PER0), we believe there is a conversation going on between the speakers and calculate the frequency.
- If there are no speakers or just 1 speaker involved in a sentence, then we say there is no conversation.
- At the end, the program calculates the total number of speakers involved in a document (as per the **<Speaker>** tag generated by Core NLP).
- Sample output of the code is displayed below, where the red box indicates that there is not conversation in a sentence and a green box indicates a conversation, speakers involved and their respective frequencies.



```
39
40          System.out.println("\nCurrent sentence :" + (temp+1));
41
42          if (nNode.getNodeType() == Node.ELEMENT_NODE) {
43
44              Element eElement = (Element) nNode;
45              for(int j=0;j<eElement.getElementsByTagName("Speaker").getLength();j++){
46                  String s = eElement.getElementsByTagName("Speaker").item(j).getTextContent();
47                  if(!s.contains("PER0")){
48                      if(!a.contains(s)){
49                          a.add(s);
50                          c++;
51                          System.out.println("Speaker " + ": " + s);
52                      }
53                  }
54              }
55          }
56          if(a.size()>1){
57              System.out.println("Conversation match between speakers ");
58              for(int k=0;k<a.size();k++){
59                  System.out.println(a.get(k));
```

```
<terminated> nlp [Java Application] C:\Program Files\Java\jre1.8.0_102\bin\javaw.exe (Sep 25, 2016, 9:13:36 PM)

Current sentence :23
Speaker : PER7

Current sentence :24
Speaker : PER8
Speaker : PER9
Conversation match between speakers
PER8
PER9
Frequency: 2
```

- Evaluated the accuracy by comparing the frequencies using manual approach. (Took few chapters from the testing book, manually annotated the conversations between speakers and compared with the program result).
- In line 21 of the code, please change the path to your system path where the input file is present. Apologies for providing the absolute path in the program.

## Advantages:

- Fully automatic approach where the program counts the number of conversations between different speakers.
- Naïve approach to calculate the conversations, so easy to write code.

## Assumptions and Limitations:

- The basic assumption of this approach is that conversations happen per sentence. That is, given a sentence, the Stanford CoreNLP generates speakers for each sentence. For the next sentence, different set of speakers are generated by the library and due to this, conversations are reported between speakers generated by the library.
- This can be a huge drawback since a conversation between 2 people can continue for many sentences. However the Stanford CoreNLP library does not keep track of the same speaker in the <Speaker> tag and hence this approach does not consider conversations beyond a sentence.

- This approach is only based on the "speaker" tag of the annotated text, and it is possible to supply the character names as a properties file which would make this approach better.
- We could not run the code for entire book due to java heap space error. Therefore only a few chapters were considered, both for development and testing.

## Approach Two:

We combed through the annotated file manually and noticed certain patterns which could help determine whether it is a conversation or not.

Using our observations, we wrote a python script which would strip off certain paragraphs that are not part of any conversation. But then, we faced issues with recognizing indirect speech sentences and we lost a lot of useful information. The script is included along with the submission. To run the file, please compile and execute findConvo.py

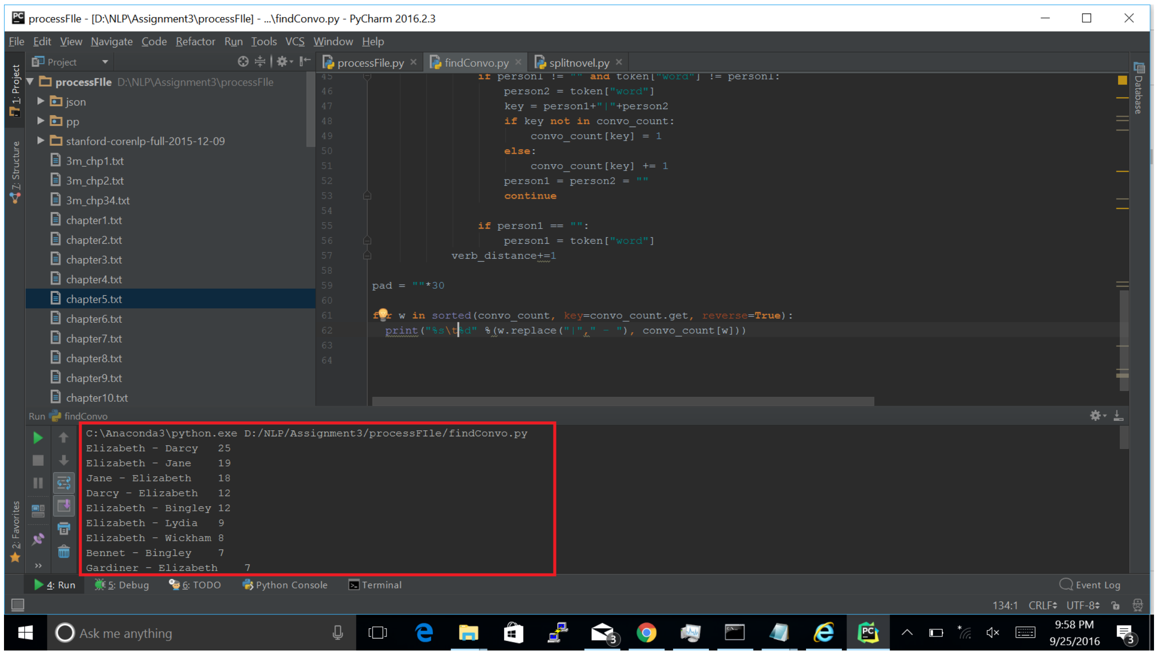
We then retorted back to annotating whole text and chose json as our output format to make our programming easier. Below are the patterns we observed and implemented as a python script to pick up conversations:

- Two people having a full conversation in a context is counted as 1 conversation.
- The proper noun mentions are outside the double quotes.
- They have pos tags NNP following or preceeding with a VBD (ex: "X said,") and named entity relation is annotated as a person.
- They have pos tags NN following or preceeding with a VBD and speaker is not the author himself.

With these assumptions, we annotated the fiction novels as a json file and passed as input to our python script. Using our above said assumptions, the script categorized blocks of the novel as a conversation or not and stored the person information and the count into a dictionary.

### Observations:

When we ran our script for "Pride and Prejudice", it was able to reasonably identify and categorize the conversations with their frequencies. It could even differentiate person1 talking to person2 and person2 talking to person1 at different circumstances.



```
                if person1 != "" and token["word"] != person1:
                    person2 = token["word"]
                    key = person1+"|"+person2
                    if key not in convo_count:
                        convo_count[key] = 1
                    else:
                        convo_count[key] += 1
                    person1 = person2 = ""
                    continue

                if person1 == "":
                    person1 = token["word"]
            verb_distance+=1

pad = ""*30

for w in sorted(convo_count, key=convo_count.get, reverse=True):
  print("%s\t%d" %(w.replace("|"," - "), convo_count[w]))
```

```
C:\Anaconda3\python.exe D:/NLP/Assignment3/processFIle/findConvo.py
Elizabeth - Darcy   25
Elizabeth - Jane    19
Jane - Elizabeth    18
Darcy - Elizabeth   12
Elizabeth - Bingley 12
Elizabeth - Lydia   9
Elizabeth - Wickham 8
Bennet - Bingley    7
Gardiner - Elizabeth    7
```

Given more time, we believe we could identify the shortcomings of this approach and further optimize the results for better accuracy.

However, when we ran the same script for "The three musketeers", the accuracy reduced. When we manually investigated the reason for this, it is because of the author's different styles of writing and addressing each character with different names based on the context and situation. Also, there were some invalid data passed as conversations. This needs to be addressed so that the script runs generically on most novels.

### Limitations:

- The script identifies conversations that fall only to the predetermined pattern.
- 100% accuracy could not be achieved with this approach.
- The success of the script largely depends on the writing style of the input.

## Given More Time:

The CoreNLP package makes use of a large amount of RAM space. When annotationg large files, we came accros many erros. Given more time we would run our make use of better computational capabilities provided by Big Red and other IU given servers. We would also test our approaches more extensively, to yeild higher accuracy and better results. The annotated file contained a lot of information showing a lot of interesting relationships amongst the words in a sentence. A deeper look into the file could have given us better insights.