Quotation Parsing and Speaker Attribution in Narrative Texts

Stephen Bradley
B.A. (Mod.) Computer Science, Linguistics and a Language
Final Year Project, April 2015
Supervisor: Dr. Carl Vogel

School of Computer Science and Statistics
O’Reilly Institute, Trinity College, Dublin 2, Ireland
Declaration

I hereby declare that this thesis is entirely my own work and that it has not been submitted as an exercise for a degree at any other university.

______________________________  April 7, 2015

Stephen Bradley
Permission to Lend

I agree that the Library and other agents of the College may lend or copy this thesis upon request.

__________________________  April 7, 2015

Stephen Bradley
Acknowledgements

First and foremost I would like to thank my supervisor Carl Vogel for his guidance over the duration of this project, and for the years he spent as our course director.

Thank you to Martin Emms for continuing Carl’s excellent work as course director.

A very special thanks to my friends and especially my classmates for helping me retain my sanity over the last four years.

Finally, I would like to thank my family. None of this would have been possible without their love and support.
Abstract

This CSLL final year project was undertaken under the supervision of Doctor Carl Vogel. The aim of the project was to create a tool for performing automatic quotation extraction and speaker attribution in narrative texts. There was an interest in creating such a tool in order to conduct research in the area of Computational Stylometry. The resulting system is SID, a naive salience-based speaker attribution system which takes Project Gutenberg text files as input, and outputs a file containing the dialogue for each speaking character from the text. Due to the general lack of research in the area of automatic speaker attribution, SID is of great value and provides a good foundation for future work.
Contents

1 Introduction 8
   1.1 Aim ......................................................... 8
   1.2 Motivation .................................................. 8
   1.3 Overview ................................................... 9

2 Problem Overview 10
   2.1 Quotation Extraction ....................................... 10
      2.1.1 Quotation Marks ....................................... 11
      2.1.2 Terminal Punctuation ................................. 12
      2.1.3 Paragraphs ............................................... 13
      2.1.4 Scare Quotes ........................................... 14
      2.1.5 Text Files ............................................... 14
   2.2 Speaker Attribution ....................................... 15
      2.2.1 Character Identification .............................. 15
      2.2.2 Character Gender Identification .................... 15
      2.2.3 Quote Speaker Dependency ............................ 16
      2.2.4 Anaphora ............................................... 16
   2.3 Background Information .................................. 18
      2.3.1 Part-of-Speech (POS) Tagging ....................... 18
      2.3.2 Named Entity Recognition (NER) ................... 19
      2.3.3 Dependency Parsing ................................. 20
      2.3.4 Hyponomy and Hypernymy ............................ 22

3 Existing Work 23
   3.1 Hierarchical Rule-based Information Extraction ........ 24
6 Conclusion

6.1 Future Work
6.1.1 Corpus Building
6.1.2 Further Testing and Error Analysis
6.1.3 Parameter and Property Settings
6.1.4 Nominal Anaphora Resolution
6.1.5 Other Languages

6.2 Concluding Remarks

References

Appendix
Chapter 1

Introduction

1.1 Aim

The aim of this project was to create a tool to be used for automatically extracting dialogue from a narrative text (presented as a digital text file) and attributing the appropriate speaker to each quotation. While there isn’t much of a commercial use for such a tool, especially considering its particularly niche use, such a tool would be very beneficial to those working in the area of digital humanities. The reason for this tool’s creation was to aid with a future study in the area of computational stylometry, regarding the strength of characterization of speaking characters in narrative texts.

1.2 Motivation

This project was undertaken with the view of using the finished software as part of a study into the area of Computational Stylometry. Stylometry refers to the application of stylistics (study of text in relation to their linguistics style) to different media such as art, music, and writing. Often, at least for analysing written texts, computational methods are implemented for performing stylometric studies which require statistical and corpus based analysis tools.
My project supervisor had previously conducted a study, addressing the problem of quantifying strength of characterization within plays (Vogel and Lynch, 2007). In this paper the writers defined a strong character as one whose speeches constituted homogenous categories in comparison with other characters. In other words, for a strongly defined character, a speech from them should be more attributable to them than to the play they are in, or to their author. For example, for a speech from the play Hamlet, by William Shakespeare, spoken by the character Hamlet, is Hamlet strongly defined enough that a system could attribute this speech to the rest of Hamlet’s speeches from the play, or to another character in the play, or another play by the author.

In order to accomplish this study a tool was needed which could facilitate the parsing of plays, presented as text files. The output of the tool being a number of files, one for each character in the play, containing the character’s dialogue from the entire play. With this, the dialogue could then be analysed for each character in order to quantify their strength of definition.

My supervisor was interested in performing a similar study, quantifying strength of characterization, but within narrative texts; novels, short stories etc. However, in order to do so a tool for extracting a character’s speech would also be needed. It was therefore the aim of my project to create a tool for processing a narrative text, extracting the dialogue and attributing each quotation to a speaker, and outputting data for each speaking character.

1.3 Overview

In this paper I outline the research behind, and development of, a speaker attribution system. I will first present the problems involved in constructing such a system, the existing work done in this area, before finally discussing the system which I have created.
Chapter 2

Problem Overview

The problem addressed in this project can be divided into two separate sub-tasks; quotation parsing, and speaker attribution. Quotation parsing refers to the process of extracting the dialogue from the text, while speaker attribution is the process of assigning each quote its correct speaker from the text. The specific problem of automatic speaker attribution is still relatively new and because of that, the amount of existing work pertaining to the topic is quite small (see Ch.3 for a discussion of existing work). For this reason it was this part of the project which required the most attention. Regardless, there are many problems involved in performing each task.

2.1 Quotation Extraction

There are numerous rules and stylistic guidelines as to how quotations should be marked and dialogue should be structured. In order to effectively extract dialogue from a text, it is necessary to understand some of these rules (a full knowledge of quotation stylistics is not necessary however). The following are a few points one should be aware of when developing a system for quotation extraction.
2.1.1 Quotation Marks

Quotations appear in narrative texts as instances of direct speech from a speaking character, or sometimes a character’s unspoken thoughts. Typically quotes appear surrounded by quotation marks (or inverted commas), often with a direct reference to the speaker expressing the quote. Sometimes however the speaker is omitted, which is called deictic anaphora (see Anaphora 2.2.4). Typical quotation use in dialogue is illustrated below in an excerpt from *Turn of the Screw*, by Henry James:

....From our end of the great brown hall we heard his step on the stair; whereupon Mrs. Griffin spoke. “Well, if I don’t know who she was in love with, I know who HE was.”

“She was ten years older,” said her husband.”

“Raison de plus— at that age! But it’s rather nice, his long reticence.”

“Forty years!” Griffin put in.”

“With this outbreak at last.”

There are a number of stylistic rules governing how quotations should be marked. However, given writers tend to have wildly different styles, it is not uncommon for writers to ignore these rules or have their own variations. For example some authors, such as Portuguese writer Jos Saramago, choose to not use quotation marks for dialogue, making extracting a character’s speech very difficult. This excerpt is taken from his novel *Blindness*:

Faltering, as if his lack of sight had weakened his memory, the blind man gave his address, then he said, I have no words to thank you, and the other replied, Now then, don’t give it another thought, today it’s your turn, tomorrow it will be mine, we never know what might lie in store for us, You’re right, who would have thought, when I left the house this morning, that something as dreadful as this was about to happen.

To accommodate this strange convention, Saramago uses verbs more often to indicate the speakers, however in the scope of this project, texts with dialogue conventions such as this are out of the question.
There are two conventions for marking quotations. The British convention is to use single quotation marks, and then double quotation marks for any quotes embedded inside the main one. The American convention reverses this standard, with double quotation marks on the outside, and any embedded quotes having single quotes. In the example below, (1) is written with the British convention, and (2) with the American one.

(1) 'Yes you did, you said “I’ve never been to Disneyland!”’ she said.

(2) “Yes you did, you said ‘I’ve never been to Disneyland!”’ she said.

These conventions should only be assumed to apply to English however. There are different conventions for other languages. Regardless, the system created for this project was designed to process English texts, and the texts used conform to the American standard of using double quotes. This means that the problem of misinterpreting an apostrophe as a single quote was not a factor.

2.1.2 Terminal Punctuation

Quotations often appear as stand-alone sentences, but they also appear embedded in narrative sentences, either as full sentences themselves, or integrated into the overall syntactic structure of the sentence. The context of a quote, as well as if its a full sentence, influences any terminal (occurring at the end) punctuation used inside the quotation.

For example, when the quotation is a full sentence, the final punctuation is inside the quotation marks. Conversely, if the quotation is integrated into another sentence, the final punctuation appears outside the quotation marks (examples taken from Copy-Editing, by Barbara Horn, p.69):

(1) She said, ‘You’ll be pleased to know that there is an easy way to do this.’

(2) You may say ‘He is economical with the truth’ but you mean is ‘He is a liar’.
2.1.3 Paragraphs

When a quotation runs over more than one paragraph, it is common to mark the beginning of each paragraph, but only the end of the last paragraph. For a system attempting to extract quotations, this creates a problem similar to that caused by having unbalanced bracketing. When parsing any type of information, be it natural language or programming languages, it is important when processing symbols that occur in pairs that each opening symbol has a closing symbol. Without a closing symbol, systems which iterate through the text will be unable to distinguish where the contents encased within the symbols should end. Having an uneven number of quotation marks for a quotation may cause errors for less sophisticated algorithms.

Specifically in relation to dialogue, there is a convention to use a new paragraph to indicate a change in speaker. Strunk and White (1999) say that ‘each speech, even if only a single word, is a paragraph by itself; that is, a new paragraph begins with each change of speaker.’ This rule isn’t strictly followed however. The example below is also taken from The Beast in the Jungle, a text where the author generally divides his speaker’s contributions into paragraphs:

...which made him, after a moment, stop again before her. “Is it possibly that you’ve grown afraid?”

“Afraid?” He thought, as she repeated the word, that his question had made her, a little, change colour; so that, lest he should have touched on a truth, he explained very kindly: “You remember that that was what you asked me long ago - that first day at Weatherend.”

“Oh yes, and you told me you didn’t know–that I was to see for myself. We’ve said little about it since, even in so long a time.”

In this example the female character ‘May Bartram’ has a contribution in a paragraph where the male character ‘John Marcher’ starts talking. Therefore it should not be assumed that a difference in paragraph will always correlate with a change in speaker.
2.1.4 Scare Quotes

The term scare quote (also known as a ‘sneer quote’) refers to a word or words which are marked as a quotation in order to signal that the word(s) are not meant to be taken literally, or understood in an ironic or atypical sense. Often they are used instead of pre-fixing the phrase with ‘so-called’ (Horn, 2008, p.68).

Justin Bieber fans, or ‘Beliebers’, need to be seen to be believed!

These quotes can be troublesome as they are marked in the same way as regular quotes even though a scare quote typically doesn’t have an attributed speaker. When creating a speaker attribution system there must be a method of either avoiding these quotes during the extraction process, or including them and later, during the speaker attribution process, identifying them as having no speaker.

2.1.5 Text Files

The texts used for this project came from Project Gutenberg, a distributor of free eBooks. Project Gutenberg offers thousands of books in multiple languages, and a number of download formats for their texts, including plain .txt UTF-8 files, which are easily read by most programming languages.

Text files from Project Gutenberg are structured in a way that closely mirrors how the text would have been laid out in it’s original hard-copy format. This means line length is dictated by line breaks using the new line character ‘\n’. Because of this, quotations which run over a line will be broken with a new line character. Preserving these breaks is a matter of preference although in the scope of this project it wasn’t necessary to preserve this punctuation. It was important however to preserve paragraph breaks. Paragraphs, as mentioned earlier, are commonly used to mark a change in speaker in the text, but more importantly, for extracting quotations, their presence within a large quotation may result in an uneven number of quotation marks.
2.2 Speaker Attribution

The core of the program outlined in this paper is the speaker attribution system. Automatic speaker attribution is a relatively new NLP task that has only really been addressed in the last few years. Below is an analysis of the problems encountered when attempting to attribute a speaker to a piece of quoted text.

2.2.1 Character Identification

Before considering how to attribute a speaker to a quote, one must first consider how characters are identified in a text. Generally a character in a text is any named individual; meaning they are identified with one or more proper nouns, such as Mary, or John O’Brien. Proper noun identification is an easy task in English in comparison with other languages such as German which capitalizes every noun (see 6.1.5 for details on creating a speaker attribution system for German). However there are still many cases where ambiguity can arise. If a word occurs at the start of a sentence for instance, it will be capitalized, making it harder to ascertain whether it is a common or proper noun. This is made even harder if the name also functions as a common noun or other part-of-speech. The example below is also taken from Turn of the Screw, and illustrates this problem. Because ‘Young’ is capitalized in this way, a system may interpret it as the surname Young.

Young as she was, I was struck, throughout our little tour, with her confidence and courage...

Having a full name, consisting of a first name, surname, and any number of possible middle names and titles, means there are many ways a character can be referred by other characters and the narrator; said John or said Mr. O’Brien. This presents some problems for a system looking to distinguish individual characters as it will need to know that John and Mr. O’Brien are the same person, and not separate characters.

2.2.2 Character Gender Identification

In addition to identifying the characters in the text, it is also necessary to identify their gender, which is a crucial part of resolving instances of
pronominal anaphora (see 2.2.4). Resolving gender is something that for the most part, humans can do automatically as we have been trained to associate names with certain genders. For a system such as this, it must either have access to some sort of name gender database, or the ability to examine the context of the name to see by what pronoun or gendered noun (husband, wife, etc.) is used to reference it, or if the character has a gendered title, such as Mr or Ms.

2.2.3 Quote Speaker Dependency

Even with the ability to detect speakers, the real difficulty with speaker attribution is trying to create an intelligent method for identifying the connection between a quote and a potential speaker in its context:

“I don’t even know why I try anymore!” David shouted angrily.

Thanks to our innate understanding of language structure and the relationship between the words in a sentence, we recognise the structural dependency between these constituents almost instinctively. In order for a system to mimic this it must be able to analyse sentences in a similar way; capable of understanding the role each word plays in the sentence and the dependencies between them.

2.2.4 Anaphora

In addition to characters having multiple names by which they can be referred to, sometimes a character will be referenced without naming them at all. Anaphora is the linguistic phenomena whereby the interpretation of an expression depends on the interpretation of another expression in the same context. What this means in practice is that an expression, some sort of noun phrase (called the antecedent) is referenced by another noun, or pronoun (the coreference) (Buring, 2005, p.1). When an expression references another, what this means is that it should be possible to replace the coreference with the antecedent without changing the meaning of the sentence.

Jess opened the chest and looked inside. “It cannot be!” she exclaimed.
In the above example, the antecedent ‘Jess’ is referenced by the pronoun ‘she’. We know that ‘Jess’ is a female name, and so even with such limited knowledge about the character we can deduce that ‘she’, a feminine pronoun, must refer to ‘Jess’. Anaphora where a pronoun replaces or substitutes a full noun phrase, is called pronominal anaphora. This along with other types of anaphora necessitate the need for some sort of anaphora resolution functionality in a speaker attribution system which can choose the appropriate character antecedent when it encounters a coreferential noun or pronoun. But this is just one type of anaphora, and is perhaps the easiest to resolve. Nominal anaphora is similar to pronominal anaphora except instead of a pronoun acting as the coreferent, it is another full noun phrase.

“What are you doing here?” Daniel asked. “That’s not your treasure!” her husband shouted.

In this example the coreferent ‘her husband’ refers to the subject of the previous sentence, ‘Daniel’. For a human reader this relationship is perfectly clear but for a computer this isn’t the case. Knowing that ‘her husband’ refers to ‘Daniel’ requires knowledge about him as a character, (his gender, role, occupation), or the ability to deuce new information about him. In comparison to pronominal anaphora, nominal anaphora demands a more comprehensive coreference resolution system (finds all expressions that refer to the same entity).

The last kind of anaphora which will have to be dealt with is deictic anaphora. Deictic comes from the word deixis, which in linguistics refers to the phenomena whereby to fully understand an expression, one requires contextual information. Deictic anaphora, in the case of speaker coreference, refers to instances of quoted text which have no explicit reference to a speaker. In order for a system to attribute a speaker to a quote, it must be able to analyse any possible dependencies between a quote and its speaker, which are usually connected by a verb in the overall sentence structure. However with deictic anaphora, these dependencies don’t exist, or at least not in the same syntactic way as they do when we have a quote-verb-speaker structure.

...I admitted however that I should like to settle down to two or three good models who would do for everything.

“Should we have often to--a--put on special clothes?” Mrs. Monarch
timidly demanded.

“Dear, yes—that’s half the business.”

“And should we be expected to supply our own costumes?”

“Oh, no; I've got a lot of things. A painter’s models put on - or put off - anything he likes.”

In the above example from *The Real Thing* by Henry James, the writer introduces the narrator as a participant in the conversation, and ‘Mrs. Monarch’ as the speaker of the first quote. After this there is no mention of them however because of the order they were introduced and the structure of the dialogue, we know who is speaking. This is known as a dialogue chain and occurs when or more speakers have their quoted speech interleaved in this alternating fashion, usually with the quotations not explicitly attributed to any speaker. It is quite obvious to a human reader that the narrator is the speaker of the second quote, then ‘Mrs. Monarch’, and finally the narrator once more. Contextual information such as the order the characters were introduced, dialogue ordering using explicit references to speakers, and even the information content of the conversation, is very easily followed by humans. For a system this is much more difficult. To carry out this task automatically one must decide what sentence information is detectable and beneficial in resolving the dependence between the quote and its speaker.

### 2.3 Background Information

In this project an assortment of different NLP tools and processes were used and in order to understand their use, it is important to have an idea of the linguistic theories behind them. For those new to the study of linguistics, this section will provide important background information on the algorithms mentioned later. But for those already familiar with the study of linguistics, this section may be skipped if so desired and referenced later when necessary; subsequent chapters will refer to this section when a new topic is introduced.

#### 2.3.1 Part-of-Speech (POS) Tagging

A word’s *part-of-speech* (POS) refers to the word’s type, be it a noun, verb, adjective, and so on. A part-of-speech tagger is a tool which attempts to
label a sequence of words with their individual parts-of-speech. Typically POS-taggers use training data consisting of sentences already labelled, to learn the probabilities that (i) a word is tagged with a particular POS, and (ii) the probability that a POS might occur after another. For example the word ‘the’ should have a very high probability of being tagged as a determiner DT (according to the Penn Treebank project POS tag set). Similarly, the probability that the tag NN (singular noun) follows DT should also be very high as nouns usually follow determiners. On the other hand more ambiguous words which may have a different POS depending on the context, will have varying probabilities for each possible POS.

POS-tagging is an important component of any NLP system as it allows the system to recognise words by type. In the context of speaker attribution POS-taggers allows a system to extract verbs, nouns and pronouns, which is important data for the speaker identification process.

2.3.2 Named Entity Recognition (NER)

In the context of NLP and information extraction, a named entity is a word or sequence of words which form the name of a real world entity, be it a person, location, company, or so on. The process of NER is that of extracting these entities from a text as well as labelling them with their type. Depending on the particular recogniser used, the number of entity types varies, but at the very least there are usually types for PERSON, LOCATION, and ORGANIZATION or COMPANY.

NER is usually broken down into two distinct problems, entity detection, and entity classification. Generally entities are extracted by identifying contiguous sequences of proper nouns with optional prepositional modifiers (Wacholder, 1997), for example ‘John Smith’, or ‘Bank of Ireland’ (‘of Ireland’ is a prepositional modifier). The classification is the part that differs widely depending on the implementation but usually involves some form of statistical learning which works by looking at the context of the entity and seeing how its context compares to that of models trained for each entity type; is the entity’s context more like that of a PERSON of a COMPANY?

In the context of the project, NER is vital for the speaker attribution task.
as it allows a system to easily find speakers by looking for PERSON entities.

2.3.3 Dependency Parsing

When looking at the phrase structure (how to group words together into phrasal units) of natural language there are two main types of structure which can be analysed, constituency structure, and dependency structure. Constituency structures refers to the type of structure created using Noam Chomsky’s phrase structure grammars. A grammar is a set of production rules for generating natural language sentences. Phrase structure rules are of the form seen below:

\[
\begin{align*}
S & \rightarrow NP, VP \\
NP & \rightarrow DET, N \\
N & \rightarrow A, N \\
VP & \rightarrow V, ADV \\
DET & \rightarrow \text{the} \\
A & \rightarrow \text{fat} \\
N & \rightarrow \text{cat} \\
V & \rightarrow \text{walk} \\
ADV & \rightarrow \text{slowly}
\end{align*}
\]

![Figure 2.1: Phrase structure grammar for generating ‘The fat cat walks slowly’, and the tree representation of this sentence](image)

Phrase structure grammars attempt to break down natural language into its constituent parts-of-speech which are connected to make larger phrasal units. Dependency structure is similar in that it too models phrases within natural language but it uses dependency relations to organise a sentence’s constituent parts into larger units.
Different types of dependency can be represented with these structures however we are only concerned with semantic dependencies. Semantic dependencies are defined in terms of predicates and the arguments given to them. A predicate can be seen as a property or characteristic of something. In Fig.2.2(a) there are a number of predicates in the sentence, the most important one being ‘walks’. The argument given to this predicate is ‘the fat cat’, as the verb ‘walks’ describes what ‘the fat cat’ is doing. The person who is performing a verb is known as the subject, in this case the nominal subject (nsubj), simply referring to the fact they are a noun.

In Fig.2.2(b) we have a two place predicate, which has a subject and a direct object (dobj). A verb’s object refers to the thing that the verb is acting upon. So in this sentence ‘The boy’ is the subject, ‘fed’ is the predicate, and ‘the cat’ is the object. The verb in both cases is the centre of the dependency structure, with the dependency arrows coming from this position. This position is called the root. When a verb is in the root position this verb is called the main verb, with any other verbs in the sentence being subordinate to it in some way.

Understanding dependency structure is important when building a speaker attribution system as in order to check any possible relations between a quote and a verb, the system must be able to check the sentence’s dependency structure to see if the quote of interest is the object of the verb. Dependency parsers are tools which can annotate sentences with their dependency structure. The one used for this project is discussed in 4.4.3.
2.3.4 Hyponomy and Hypernymy

Semantic fields are groups of words whose meanings are closely interrelated (Gao and Xu, 2013). Hyponomy and hypernymy refer to certain ways in which humans create these semantic fields. Hyponyms in a set share the characteristic of being of the same type, this type being their hypernym. So for example blue, red, and yellow are hyponyms of colour, or one can say colour is a a hypernym of blue, red and yellow. A hypernym is a thing whose semantic field encompasses many things which have a much narrower or more specific semantic meaning.

This is an important concept to understand for this project because of the importance of identifying expressive verbs. Humans can immediately identify an expressive verb; expressive in terms of whether or not it can be used to convey speech. In order for this to be done automatically, there must be way to classify a verb as expressive. Rather than simply having a list of expressive verbs, this hyponomy-hypernymy relationship can be analysed by seeing if a potential verb is a hyponym of a verb such as 'say' or 'utter'. WordNet which is an online lexical database, structured in terms of the semantic relationships between words; the different semantic fields they belong to. In 4.3.2 a library is discussed which can implement this WordNet database so that it can be used to identify these relations between words.
Chapter 3

Existing Work

Speaker attribution systems are still relatively new NLP tools and so there isn’t a wealth of existing work in the area. This gave me the opportunity to develop what would hopefully be a somewhat original approach to tackling the problem. My initial plans were for a system which would first iterate through the text and resolve only quotes with direct speaker references. It would do this by finding speech verbs for these quotes, and using a dependency parser, find the subject of the verb. On a second pass the system would then resolve quotations that had pronominal anaphoric references. It would resolve the anaphora by choosing the last mentioned character of the appropriate gender. Lastly, it would then fill in the blanks for the other quotations whose speaker was not mentioned; deictic anaphora. It was a mystery to me however exactly how this ‘filling in the blanks’ would work.

Many of the authors who developed speaker attribution systems had similar intuitions, insofar as they recognised the importance of the relationship between a quote and a related expressive verb, if one existed. However the way they extract this information differs greatly. None of the papers I studied explain a method for quotation extraction, the reason for this I assume, being that it is a task which only has one solution; using regular expressions. I was able to analyse the code from Lynch and Vogel (2007) which confirmed my thoughts on this.

Next I will discuss three papers which describe different methods for speaker
Glass and Bengay published ‘Hierarchical Rule Generalisation for Speaker Identification in Fiction Books’ in 2006. Information extraction refers to the process of extracting information from unstructured or semi-structured machine-readable documents (as opposed to documents with structured data such as XML or JSON). Rule-based information extraction makes use of rules as a means of matching patterns in data so that certain parts of the data can be extracted. These rules can be made by hand but machine learning techniques are generally used to create these rules automatically. In this paper, Glass and Bengay present a method for taking pre-existing patterns from a training set and generalising them to create a concise set of rules which can each be used for finding the speech verb, actor, and speaker, in a sentence matching the syntactic category of the rule.

3.1 Hierarchical Rule-based Information Extraction

Figure 3.1: The figure on the left is a rule for finding the Actor in a sentence. The figure on the right is a rule for finding the Speaker in a sentence.
The patterns used in their technique have a hierarchical structure with multiple levels of abstraction as seen in Fig.3.1. With a multitude of patterns such as this, generalisation attempts to reduce the number of patterns so that a single rule can apply to a number of different sentence types. When iterating through the chosen document, there is a trigger every time a quote is encountered. Three types of rules are used in order to try and match the speech verb, the actor, and finally the speaker for the target quote. So if a rule for finding a speech verb matches to the target sentence, the system can immediately look to the verb node indicates the expressive verb.

For tackling the problem of character identification they refer to different methods of Named Entity Recognition (NER). In this paper however, while identifying the need for NER in order to perform speaker attribution automatically, Glass and Bangay opted to use a manually created character list which speakers could be selected from, in order to remove the possibility of any error introduced due to NER.

While the idea of using these hierarchical rules is quite effective (speaker identification accuracy of 91.97%), I thought the system sounded a bit restrictive and it seemed to me as if it would be difficult to implement additional functionality (such as the ability to glean information from the quotations themselves which could be used in the attribution process). Similarly I wasn’t convinced that this task required the use of training data or complicated machine learning techniques. Furthermore I was sure gathering training data for the task wouldn’t be possible.

### 3.2 Nave Salience-based Identifier

In 2007 Glass and Bangay’s published ‘A nave, salience-based method for speaker identification in fiction books,’ their second attempt at speaker attribution. A nave method is one which does not make use of any complex machine learning algorithms or techniques which draw on a knowledge base. Salience is a word for a quality or feature and refers to whether something stands out from its surroundings. In the context of the paper it refers to methods which use a scoring system for choosing a word from a set of possible candidates, i.e. the word with the highest salience. This system, in
comparison to the writers’ previous one, does not use machine learning and
gives purely based on contextual information, which I think is a better
eulation of how a human may perform the task, which is why I found the
most inspiration in this paper.

The methodology behind this system is similar to the hierarchical rule-based
information extraction method in that for every quote the system searches
for an expressive speech verb involved in conveying the quote, then the sub-
ject argument of this verb, called the actor, before finally performing speaker
resolution. In order to find the verb, the system searches the quote’s context
for candidates. For stand-alone quotes which are not embedded in a narra-
tive sentence, the previous and subsequent sentences surrounding the quote
are searched. If the quote is embedded in another sentence, this sentence is
searched. These verbs are then scored based on a number of features; the
highest scoring verb said to have the largest salience, and therefore being the
most likely expressive verb for the quote. The following features are used
when scoring verbs:

- Main verb: The main verb in the sentence is awarded salience (see 2.2.3
  for a definition of ‘main verb’).

- Hypernym: A verb is awarded salience if it is a descendent of the verbs
  communicate, verbalise, or breathe in terms of a hierarchical lexical
tree.

- Adjacent sentence: A verb is awarded salience if it is found in the
  sentence in which the quote is embedded.

- Quote proximity: Verbs which occur in sentences preceding and follow-
ing the quotation are given salience depending on their distance to the
quote (larger distance, smaller salience).

Verbs which are awarded one of these features are given 1 point, with the
exception of the proximity feature which awards a number between 0 and 1
depending on the distance. The verb with the highest salience is chosen as
the most likely speech verb.

Choosing an actor works in a similar fashion. Nouns are extracted from
adjacent sentences, filtering out certain ones such as titles (Mr, Mrs). These
candidates are then scored for a number of features, the highest scoring one being chosen as the actor.

- **Subject or Object**: The candidate is awarded salience if it the subject or object of the main verb.
- **Part-of-speech**: A candidate is awarded salience if it is a noun which is person-like (*man, woman, husband*, etc.) or if it is a pronoun.
- **Proper Noun filter**: Words which are proper nouns are awarded salience.
- **Expressive verb proximity**: Salience is awarded depending on a candidate's distance from the speech verb.

If the actor chosen happens to be a direct reference (proper noun), that actor is matched to a character from their pre-prepared character list (again they did not use NER). In the case of anaphora, be it pronominal, nominal, or deictic, a speaker resolution process is used. This method uses a number of variables to reason as to the most likely speaker; whether or not an actor was found, the gender of the actor, the last person to speak, the last speakers mentioned by name and the order they were mentioned. Glass and Bangay created a hand-coded decision process which is shown in Fig.3.2.

In comparison to their first attempt I felt this method aligned more with my initial methodology for tackling the problem.

### 3.3 Rule-based and Statistical Learning

The third system I investigated was described in ‘Automatic Attribution of Quoted Speech in Literary Narrative’, published by Elson and McKeown in 2010. It too took inspiration from Glass and Bangay’s salience based method but also makes use of statistical learning techniques. Again, while I found their solution to the task quite interesting, I wanted to avoid any methods involving machine learning as I did not have access to the necessary corpora.

Elson and McKeown describe their system for attributing a speaker to a quote as a three step process. First, there is a preprocessing step to identify
all named entities and nouns which appear in the text preceding the quote (strangely, they do not search text after the quote). These are the candidate speakers, similar to the actors from Glass and Bangay (2007).

The second step is to classify the target quote into one of a number of syntactic categories. These categories describe the placement of a quote within a sentence, or overall discourse structure. These categories and their occurrence rate in the corpus Elson and McKeown analysed is shown in Fig.3.3.

![Diagram](image-url)
Classifying quotes into these categories serves to group together instances where the syntax strongly suggests the identity of the speaker. Where a quote’s category strongly implies a speaker, this character is chosen without any need for further help. These categories are Added quote, Apparent conversation, Quote-Said-Person trigram, and Quote-Person-Said trigram.

The third and final step is to create a feature vector for each candidate speaker for the target quote. A feature vector is a list of values. Every position in the vector relates to some characteristic feature, such as the candidate’s proximity to the quote, etc. The system uses three probabilistic models created using a number of learners, each learner trained on feature vectors for thousands of quotes from their corpus. Each of these models is for resolving a different quotation category; No apparent pattern, Quote alone, and Anaphora trigrams (categories whose structure do not strongly imply a speaker). During the attribution process, if a quote falls into one of these categories the feature vectors for the speaker candidates are sent to the appropriate model, which calculates for each candidate the probability that he/she is a speaker. The candidate with the highest probability is attributed to the quote.

Elson and McKeown, in comparison to Glass and Bangay (2006, 2007), did not rely on preprepared character lists for their system. They created their own system for named-entity recognition and a character nominal extractor for finding candidate characters. They process their text with the Stanford NER tagger (see 4.3.3 for more information on Stanford NLP tools) and extract contiguous proper nouns (names which consist of more than one word). Their system groups together proper names which reference the same character so the name ‘Mr Sherlock Holmes’ is identified as belonging to the same group as ‘Mr. Holmes’, ‘Sherlock Holmes’, ‘Sherlock’, and ‘Holmes’. In order to handle character nominals, such as ‘the fat driver’ or ‘her husband’, they use a regular expression that matches a determiner, any modifiers (adjectives), and a head noun. They do not give explicit detail as to how they perform coreference resolution between such character nominals and named characters, but I assume their system does this as part of the third step in their process.

Elson and McKeown also explain their encoding, cleaning, and normalizing processes they perform on texts before speaker attribution. They mention...
that this step is not performed on the text as whole, but rather on the text between a candidate speaker and the target quote. The normalizing steps are as follows:

- Replace quote and character mentions with symbols.
- Replace expressive verbs with symbols.
- Remove extraneous information such as adjectives and adverbs.
- Remove sentences and paragraphs where no quotations, pronouns, and names appear.

### 3.4 Review

All three papers offer very different methods for solving the speaker attribution task. It was important for me to follow my own intuitions in regards how the system should function and looking at this material, it was obvious that the methodology Glass and Bangay (2007) aligned most with my ideas. Regardless, all of the papers offered helpful information that aided me in developing my system. The following are the features I took from my research which I hoped to integrate into my system:

- A nave system which does not make use of machine learning. Rather it will use salience-based scoring for identifying an expressive verb and actor.
- Quotations will be extracted using regular expressions.
- Quotations will be replaced with symbols before performing the attribution process so that the dependency relation between an embedded quote and its host sentence will be easier to detect.
- The salience-based scoring for both verbs and actors will use a number of features based on the quote’s contextual information.
- NER will be used for identifying named characters. Contiguous names identified by NER will be grouped together to form character sets. Nominal characters will be identified using regular expression.
• The gender of characters will be ascertained via the context in which they appear and how they are referenced by other characters, as well as by the titles they have, if any.

• The eventual speaker attribution process will use a decision tree to make its final decision. The decision tree will take into account information such as the identity of the last speaker, whether or not an actor was identified, and if the target quote is in the last paragraph as the previous.
Chapter 4

SID - SpeakerIDentifier

In this chapter I will first give a brief overview of the overall methodology behind my speaker attribution system (called SID, for ‘Speaker Identification’) and the components necessary to complete each part, discuss the resources and tools used during the project’s development, and finally the architecture and algorithms that make up SID in its current iteration.

4.1 Background

An important goal for this project was to create a system that, in addition to using no machine learning, functioned in a way that somewhat emulates the human speaker attribution process. While I can see the advantages of using more mechanical statistical learning algorithms, often I find that they work in a fashion completely unlike that of how a human might solve the task. This is of course understandable as computers generally don’t think like people. Humans are good at reasoning problems which require an understanding of the meaning or semantics, computer’s on the other hand are good at solving problems which require brute force computational power.

Whilst researching this task I consciously paid more attention to how I was reading a narrative story as I was reading. I realised that we make use of faculties that at the moment are not easily emulated by computers. Our
ability to understand what is happening in a text informs us greatly as to who the speakers are, and often allows us to predict who will speak next.

I did notice however that even with authors who have quite a uniform style of writing, it is sometimes necessary to look back at what you have just read, to try and figure out who is currently speaking, either through a loss of concentration or perhaps some ambiguity introduced by the author. When you lose the narrative flow, it is sometimes necessary to take a more analytical approach to figuring out who the current speaker is; looking at how spoke previously, who is speaking after the target quotation, and sometimes with which character you would associate the contents of the quote (my system nor do others I have analysed take into account the contents of a quotation during the speaker attribution process). It is for this reason that I think creating a system which relies only on the text being analysed to inform its attribution process, is a good method for tackling the problem, and is what I have tried to capture in my system.

4.2 Methodology

SID functions similarly to the system outlined by Glass and Bangay (2007). That is to say for every quote, it tries to find an expressive verb, an actor associated with this verb, and finally resolve the speaker. Before discussing the details of SID’s inner workings, I will first discuss the resources and tools that went to its creation.

4.3 Resources and Tools

4.3.1 Python

SID was written in a language called Python. Python is a high-level, general purpose programming language that has become extremely popular in the last few years and is one of the most widely used languages today. The creators of python have a philosophy for simplicity and readability, which means python is very concise and quite understandable for people who may not be familiar with the language. In the past, Python has allowed me to
create programs very rapidly. This is due in part to its concise nature. With Python, a lot can be said with very little code. Without going into too much detail about the technical aspects of Python, one of its main strengths over a language such as Java, is that it allows for very quick prototyping and in most cases vastly increased productivity. While Python caters for object-orientated programming (and other paradigms such as imperative and functional), it is not enforced. While I do agree that an object-orientated paradigm can be beneficial, in that it produces well structured code, I also find OOP very slow and restrictive. With this project I knew that I would constantly be changing the architecture of my system as I tried new algorithms and methods for tackling each of the sub-tasks involved. Python allows for rapid changes to be made very quickly without the need for fiddling with separate class files (which are not necessary for Python, everything can be stored in one file if so desired).

Aside from the benefits to work pace, Python is also a language that is suited for NLP, with its attractive methods for working with strings (slicing operations) and lists (functional programming style list comprehensions). It has a powerful regular expression module and there are many libraries available for NLP tasks, including the Natural Language Toolkit (NLTK).

4.3.2 NLTK

The Natural Language Toolkit is a suite of tools designed for use in natural language processing applications. It includes vast array of tools as well as numerous corpora for training and testing. For this system one module of importance from NLTK was the WordNet library. WordNet is an online lexical database for English created at Princeton University. WordNet organizes words into what they describe as sets of cognitive synonyms, or synsets. A synset for a word contains a reference to each of the different senses or meanings of the word.

These synsets are organized into semantic fields (see 2.3.4), which represent cognitive relationships. For example, words can be organized into the synonym relationship which relates words which are similar enough in meaning that often times they can be swapped without affecting the meaning of the sentence. Words are also grouped according to the hyponomy-hypernymy relationship. WordNet is structured in a way that links synsets together
using these relationships. This is illustrated in Fig.A.1, where the different meaning of the word ‘shout’ are explained, and its synonyms and hypernyms are provided.

NLTK was an important tool in this project as it gave SID the functionality it needed to reason if a verb was particularly expressive or if an actor candidate was person-like.

### 4.3.3 StanfordCoreNLP Tools

The natural language processing group at Stanford University have released a number of NLP tools over the years and their core tools integrates these together into a single library which can easily be integrated into any system. In order for my system to work, it needed to be able to analyse every sentence in the document for information such as part-of-speech, dependency structure, and named-entity data. Core tools has the capability to provide this functionality and more. The only problem with using these tools however is that they are written in Java and intended to be used in Java programs. People have created wrappers which allow the tools to be used use in other languages, including one for Python, however at the time of creating this project, not all of the tools in the core suite were supported in Python.

It is possible however to run the tools via the command line. My system calls Core tools via a Java command and passes it the file to be processed as an argument. When called in this way the Core tools operate as a pipeline of annotators, each one requiring the output of the previous one. Running the Core tools does take quite a long time (around one minute for an average sized novel), however efficiency was not a concern for this iteration of SID. When the tools finish annotating the document the results are printed to an XML file. The XML tree structure of an output document from Core tools

```python
>>> from nltk.corpus import wordnet as wn
>>> print(wn.synsets('man'))
[Synset('man.n.01'), Synset('serviceman.n.01'), Synset('man.n.03'), Synset('homo sapien.n.02'), Synset('man.n.05'), Synset('man.n.06'), Synset('valet.n.01'), Synset('man.n.09'), Synset('man.n.09'), Synset('man.n.10'), Synset('world.n.06'), Synset('man.v.01'), Synset('man.v.02')]
```
is shown in Fig.4.2.

4.4 Methods and Algorithms

SID’s main method can be split into main parts pre-processing, and speaker attribution. The steps involved in pre-processing include, reading the text file, removing extraneous information, extracting quotations, replacing quotations with numbered symbols, and finally identifying characters. The speaker attribution process functions whilst iterating through the text line-by-line. At all times the characters involved in the scene are tracked. When a quote is detected, verb extraction and scoring occurs, noun extraction and scoring occurs (to find the Actor), and finally speaker resolution with a decision tree is used to choose the most likely speaker.
4.5 Preprocessing

4.5.1 File Cleaning

In addition to the story and contents books usually contain additional information such as publisher details. Project Gutenberg eBooks contain this data and more in their header and footer. Generally the header contains information about the language the text is presented in, the release date, and character encoding. The footer contains the product license and information about distribution policy and terms of use. All of this information, in the scope of the task, is extraneous and would only serve to cause possible error during the attribution process. To avoid this files are cleaned of as much of this extraneous information as possible before any further actions are taken.

In most cases the header and footer are indicated by some text with a number of asterisks indicating where they end and start respectively (Fig.4.3). Some texts do not comply to this format but enough texts do that it is possible define a method for removing them. SID uses regular expressions to replace
The function works by first concatenating all of the text from the file into a single line. This allows the entire text to be searched with a regular expression at the same time. At every paragraph break or point of concatenation, a symbol is added, ‘˜newline˜’. This is to avoid losing paragraph breaks when the novel is split into its original lines again. To remove the header a regular expression is compiled which is used to replace all of the text before and including the header.

```
header_regex = re.compile(r'[^*]+*?START OF (THIS|THE) PROJECT GUTENBERG (EBOOK|ETEXT)(\w| )+ *?\*+')
novel_string = re.sub(header_regex, '', novel_string, count=1)
```

Similarly, to remove the footer a regular expression is compiled which replaces the footer and any text after it with the empty string. After the header and footer are removed, the novel string is split at every `newline`, into its original lines.

### 4.5.2 Dialogue Extraction

After the chosen text has been cleaned, the dialogue can then be extracted. Due to the uniform nature of quotations, they can be very easily located using regular expressions. The text is first concatenated into a single string ensuring multi-line quotations are captured as one quotation. Again ‘˜newline˜’ symbols are inserted at concatenation points to preserve paragraph breaks. After extracting the quotations using a regular expression, they are stored in a list, thus preserving the order they were found in. This is important later when the quotes are substituted with numbered tags.

```
# compile regex for identifying quotes
quote_regex = re.compile(r'("[^"]*")')
# gather dialogue chunks for later indexing
for dialogue_chunk in re.findall(quote_regex, novel_string):
    dialogue_chunks.append(dialogue_chunk)
```
At this point it is worth noting that my system does not exclude scare quotes when extracting quotations. They are extracted just like any other regular quotation. I decided the best way to handle these is during the speaker attribution process and not during the extraction process. It is also worth noting that at this point my system cannot handle texts which contain an uneven number of quotation marks. Where this occurs it is either a mistake in the text, which I encountered in many Project Gutenberg texts and corrected myself, or is an instance of quotations running over multiple paragraphs. None of the texts tested had quotations of this type so a solution to handle these types of quotation has yet to be implemented.

4.5.3 Quotation Substitution

In order to make the process of analysing sentences containing quotes easier, quotations are substituted with tags \(<\text{QUOTE}\>\). This is done firstly to avoid mixing up quoted and narrative text when extracting nouns and verbs during the attribution process. Secondly, the dependency between a quote and a verb is easier to detect if the quote itself is represented as a single noun.

(1) “What are you doing here?” she shouted, as she crossed the room, “You are no longer welcome here!”

(2) \(<\text{QUOTE}\>/> she shouted, as she crossed the room, \(<\text{QUOTE}\>/>!

(3) \(<\text{QUOTE}\>, she shouted, as she crossed the room, \(<\text{QUOTE}\>/>!

Figure 4.4: (1) is the original sentence containing embedded quotations, which themselves are full sentences with terminal punctuation. (2) is this sentence with the quotations replace with tags. Note the punctuation has been moved outside. The ’?’ after the first quote is problematic for sentence splitting, so it is replaced with a comma in (3).

In order for SID to identify the connection between a quote and a verb expressing it, it must analyse the dependency structure. In Fig.4.5, the entire quotation can be seen as the noun object \(<\text{QUOTE}\>/>of the verb uttered. By replacing the quotation with a tag, the \(<\text{QUOTE}\>/>is labelled as a noun by the POS-tagger, making it easier for the dependency parser to detect its relation to the verb. Lastly, replacing all of the quotations with single words
removes a lot of the text that the Core tools would otherwise have to unnec-
essarily analyse.

There are many steps involved in replacing quotations with these tags. The system must ensure that any terminal punctuation in the quotations won’t get removed when the quotations are substituted. Therefore this punctuation must be identified, copied, and then placed outside of the quotes containing them. To do this the novel string is chunked into narrative and quoted text. Doing so allows SID to easily find the quotation chunks and move any terminal punctuation outside the quotation marks. After doing this the chunks are concatenated into a string again.

Next, every quotation is substituted with a <QUOTE_> tag, using the regex substitution method. The regular expression used for doing this is the same as the one for matching quotations. After replacing the quotations, the tags are numbered so that they can later be used to reference the quotation they refer to, <QUOTE_n>. This form of substitution requires a pattern which has a counter that increments for every match, and this is not possible using regular expressions so it was necessary to do this iteratively. The novel string is again chunked into narrative and quoted text in the same way as before. The chunks are then iterated through, so that every <QUOTE_> can be processed. When a <QUOTE_>is encountered the system checks if there is any terminal punctuation and processes the tag based on this.

```python
for i in range(len(novel_chunked)):
    # quote followed by .
    if re.match(r'<QUOTE_>\.', novel_chunked[i]):
        novel_chunked[i] = "<QUOTE_" + str(quote_count) + ">."
        quote_lookup_table[novel_chunked[i][0:-1]] =
            dialogue_chunks[quote_count]
        quote_count += 1

    # quote followed by ?
    elif re.match(r'<QUOTE_>\?', novel_chunked[i]):
        # check if quotation is embedded within other sentence
        if (i+1) < len(novel_chunked)-1 and
            novel_chunked[i+1].strip()[0].islower():
```
When a tag with a terminal ‘?’ or ‘!’ is encountered, these are sometimes replaced with a comma. This is done in order to avoid any errors later when performing sentencing splitting. How these errors arise is shown in Fig.4.4.

After replacing all of the original tags with their indexed equivalents, they are added to a dictionary as keys, with the values being the quotations they refer to. For example, the 7th `<QUOTE>` tag is replaced with `<QUOTE)))), and this tag is used to refer to the quotation which in the 7th position in the list of quotations compiled earlier. The process of replacing a tag that ends with a ‘.’ and a ‘?’ is shown above in order to contrast the additional work needed to deal with question marks that may disturb the dependency structure of a sentence when moved outside their quotation marks.

### 4.5.4 Paragraph Preservation

The next step therefore is to record paragraph information; what sentences belong to the same paragraph. The novel string is first divided into paragraphs; paragraph breaks represented by “newline”. These paragraphs are then divided into sentences using a sentence tokeniser from NLTK. A counter is incremented to index each sentence. For each sentence in the paragraph its index is added to a set representing the paragraph. This set is then stored in a dictionary, where every key is a sentence index, and every value is the full set of sentence indexes constituting the paragraph.

After this the novel string is split into sentences using the tokeniser and each sentence is scrubbed of trailing or leading whitespace, any “newline” symbols are replaced with a space, and any empty strings created in the splitting process are removed. These processed sentences are then ready to be analysed by the Core tools.
4.5.5 Running StanfordCoreNLP

At this step the chosen text has had its header and footer removed, quotations replaced with numbered tags, and has been divided into a list of sentences. These sentences need to be analysed using StanfordCore NLP in order to obtain token information on their part-of-speech and named-entity type, as well as the sentences’ dependency structure. In order for the Core tools to analyse the sentences from the text they must first be printed to a file which can be passed to them. The Core tools are configured with a properties file, the layout of the one used for this system is shown below:

```properties
annotators = tokenize, ssplit, pos, lemma, ner, depparse
_tokenize.options.tokenizeNLs = true
ssplit.eolonly = true
```

The first line indicates the annotators to use. The other two lines of code configure the system to split sentences only at new lines. This means that no further sentence splitting is performed ensuring consistency between the NLTK sentence splitter and the Core tools splitter. To accommodate this, when the sentences are printed to a file to pass to the Core tools, a new line is printed between each one.

After annotating the data, which can take up to a minute for a large text, the results are printed to an XML file which my system can then read using Python’s XML processing library. The document is stored in a tree data structure which can be accessed using the XPath query language. XPath statements operate on this tree structure in order to access specific nodes of data. Complex statements can be created with arguments and conditions in order to access specific nodes or to search for sets of nodes that match certain criteria. The code below is used during the speaker attribution process for first retrieving the sentence node corresponding to the sentence in the XML document at position sentence_pos+1, then retrieving both token data and NER data for the tokens. The reason for the the +1, is that XML indexes start from 1, not 0, and so any list index used to identify an object in the XML file must be incremented by 1.
4.5.6 Character and Gender Identification

Before performing the speaker attribution process, SID first scans the text in order to identify any characters and to create alias sets for them. An alias is a name which they are referred to by the narrator or other characters. Often a character has many aliases which need to be grouped together into an alias set which represents the character as a whole. The character alias sets for the text *The Beast in the Jungle* is shown below:

```
{‘John Marcher’, ‘John’, ‘Marcher’}
{‘Miss Bartram’, ‘May Bartram’, ‘May’, ‘Bartram’}
```

SID does this by iterating through every sentence in the text and checking if the NER data for any of the tokens contains the type PERSON. If this type is found, the system iterates through the sentence and gathers together tokens which form a contiguous alias. The new alias is then sent to a function which adds it to an existing character’s alias set, or a new character alias set is created to which it is added. This function works by checking the components of the new alias to see if any of these names already belong to an existing alias set. If so this new alias is added to the set as a string. This means if the alias already exists in the set it will automatically be removed as every element in a set must be unique. If no existing alias set contains any component of the new alias, a new alias set is created and the alias is added to it. In addition to this, if this new alias consists of more than one word, its many permutations are also added to the set. For example, if the alias ‘John Marcher’ is found to not belong to any existing alias set, a set is created containing all of its variations, {‘John Marcher’, ‘John’, ‘Marcher’}.

While this method works to, in most cases, retrieve all of the speakers, it
also retrieves any other people that are mentioned by name who might not who have a role in the text, and any other entities that the NER tool might incorrectly label as a PERSON. For example, the full list of retrieved entities for *The Beast in the Jungle* is shown below:


Some of these named PERSONs (‘David Price’ and ‘Martin Secker’) are the names of publishers and other people involved in the creation of the text. A more comprehensive file cleaning step which removes more extraneous information would prevent such erroneous characters from being extracted by the system. SID remedies this by asking the user to filter out incorrect character entities. SID also asks the user to assign each remaining character a gender.

In order to identify a narrator, SID searches the text for any mentions of the words ‘I’ or ‘me’ outside of quotations. If it finds one of these words, it will automatically add the ‘NARRATOR’ to the list of characters. If incorrectly identified, the user can remove the narrator when filtering out other characters.

### 4.6 Speaker Attribution

#### 4.6.1 Relevant Character Tracking

In order to resolve pronominal anaphora a speaker attribution system is necessary to keep track of the characters involved in the current scene. SID does this by maintaining a priority list of tuples for each character, containing a character’s alias set, and a score representing their relevance, which is reordered as characters are mentioned. During the attribution process while SID iterates through the text line-by-line, each line is searched for mentions of the PERSON entity. If a PERSON type noun is found, SID attempts to match it to its corresponding alias set in the priority list, if one exists. If the entity wasn’t filtered out by the user, the alias set is then promoted to the top of the list with a score of 1.0 while the other character’s alias sets are demoted by a small amount (multiplying their current value by 0.9).
method of character tracking was based on the method outlined in Glass and Bangay (2008).

4.6.2 Verb Scoring

The first part of the speaker attribution process is finding possible expressive verbs which link a character to a quote. SID uses a method of scoring verbs based on features similar to that used in Glass and Bangay (2008). The scoring method is initiated every time a quote is detected in a sentence as SID iterates through the text. The scoring process differs depending on the quote’s context. If the quote is embedded in a sentence, this host sentence is analysed, verbs are extracted and their positions are added as keys to a dictionary, which will hold their scores (initially set as 0). If the quote is stand-alone, the sentences preceding and following the quote are analysed and verbs are extracted (unless they themselves are stand-alone quotes, in which case they are not searched). The verbs are then scored based on a number of features. All of the features award a verb a score of 1.0, apart from the distance measure which gives a score between 0 and 1. After scoring, the verb with the highest salience is chosen and its location is stored for use during actor scoring (list indexes are used when recording verb or noun choices in case of duplicate words).

Main verb: The main verb in the sentence is awarded salience. SID does this by extracting dependency structure information on the target sentence from the XML document using an XPath expression. The code below shows how a sentence node is retrieved and the root dependency node is extracted from inside this:

```python
sentence_path = './document/sentences/sentence[@id="'' + str(sentence_pos+1) + ''"]'
sentence_node = novel_root.find(sentence_path)
root_dependent_node = sentence_node.find('./dependencies[@type="collapsed-ccprocessed-dependencies"]/dep[@type="root"]/dependent')
```
The condition @type="root" ensures only the root node is retrieved. It is at the root position where we usually find the main verb (but not always), and so with this node the system checks if the index of the root word is one of the candidate verbs. If so the verb is awarded a point. If no verb is in the root dependent position, the verb closest to the root is scored for this feature. In the case of the quotation being embedded in a sentence, only one verb should be scored for this feature, if any. For stand-alone quotes, there may be one main verb per sentence searched.

**Expressive verb**: A verb is awarded salience if it is expressive. The WordNet library from NLTK is used to check this. This is performed by seeing if one of the verb’s hypernyms is contained in a set of expressive verbs. This is based on the method used in Glass and Bangay (2008), however I use a larger set of expressive verb for more coverage:

```
```

SID also checks the WordNet definition of the word to see if the description contains words such as ‘verbal’ or ‘declare’. I noticed a marginal improvement however more testing is necessary to see to what extent checking the definition helps. The function for checking the expressivity of a verb is shown below.

```python
def is_expressive_verb(verb):
    exp_verbs = ["say", "verbalize", "communicate", "state",
                 "declare", "complain", "utter", "talk", "express"]
    syn_set = wn.synsets(verb, pos=wn.VERB)
    hyp_list = []
    is_expressive = False
    if len(syn_set) > 0:
        for sense in syn_set:
            hyp_list.append(sense)
            hypr = sense.hypernyms()
            hypr.extend(sense.root_hypernyms())
            for h in hypr:
                hyp_list.append(h)
    for h in hyp_list:
```

46
The first part of the algorithm iterates through the synset for the target verb, retrieves the hypernyms of each of the verb’s senses, and stores them in a list. If the definition of any of these hypernyms contains the words ‘verbal’ or ‘declare’ the function returns true. The function then checks if any of the collected hypernyms are in the set of expressive verbs, and if so, the function returns true. More than one verb can be scored for this feature.

**Verb-quote dependency**: If the target quote is the direct object of the verb, the verb is awarded salience. This function also requires information on the dependency structure of the target sentence from the XML file. If the position of one of the candidate verbs stands in the governor position, in relation to the quote, then the verb is awarded salience. This feature can only apply to one verb, in the same way as the main verb feature. Note that in the case of stand-alone quotes, there can be no structural dependency between the target quote and any verbs as they are in different sentences. Therefore this feature is not used for stand-alone quotes.

**Quote distance**: A verb is awarded salience depending on its distance from the target quote. In order to calculate the salience awarded, the maximum distance is first found by finding the verb candidate furthest from the quote (in terms of how many words are between the verb and the quote). The salience for every verb is inversely proportional to their distance from the quote divided by this largest distance, normalised between 0 and 1.

```python
def distance_from_word(word_scores, target_word_pos):
```
```python
word_distances = { w_pos:0 for w_pos in word_scores }
total = 0
for w_pos in word_distances:
    word_distances[w_pos] = abs(target_word_pos - w_pos)
total += abs(target_word_pos - w_pos)
for w_pos in word_scores:
    dist_score = 1 - (word_distances[w_pos]/total)
    if dist_score != 0:
        word_scores[w_pos] += dist_score
    else:
        word_scores[w_pos] += 1
return word_scores
```

### 4.6.3 Actor Scoring

After choosing the most salient expressive verb, SID tries to find the noun which is related to this verb, and is therefore the speaker of the quote. The process for choosing an actor is the same as for selecting a verb. First, all candidate nouns are extracted from the quote’s context. All non-nominative pronouns are removed as only nominative pronouns may stand in the subject position of a verb. These pronouns are removed by first identifying them using their part-of-speech, and then checking if they are in a set of nominative pronouns. Titles (Mr, Mrs, Dr, Prof) are also removed by checking their presence in another filtering set. Finally, if the `<QUOTE_n>` object itself was gathered as a candidate, it is immediately removed. After filtering out these words, the candidates are scored. As with the verb scoring system, all features award a score of 1, except the distance measure which is a floating point number between 0 and 1.

**Person-like**: A candidate is given salience if it is person-like. Using WordNet, the system checks if the word ‘person’ is in the candidate’s hypernym hierarchy. This ensures words such as man, woman, or husband, are awarded salience and not words such as chair or computer. The function for deciding if a noun is person-like works similarly to the one for deciding if a verb is expressive.
Personal nominative pronoun: If a candidate is a personal nominative pronoun it is awarded salience. This is easily checked by seeing if the pronoun is in the set of nominative pronouns:

\{'I', 'you', 'he', 'she', 'it', 'we', 'they'\}

However, if the pronoun is ambiguous then the system must perform additional checks. “You” and “it” are considered ambiguous because their form does not change depending on whether they are in the nominative (subject) or accusative (object) case. In order to decide what case they are in, the system checks if they are in the subject or object position of a verb in the sentence. If they are the subject it means they are nominative and can be awarded salience.

Person entity: If the word is a PERSON according to the named entity recogniser, then it is awarded salience. This is checked by retrieving the NER tag for the sentence and checking if the tag at the same position as the word is of type PERSON.

Subject position: If the candidate is the subject of the expressive verb found in the previous verb scoring step, then it is awarded salience. Sometimes however, a candidate may not be the subject of the verb from the previous step, but a verb that is related to it via a verb chain. A verb chain occurs when a number of verbs are used in sequence that all relate back to the main root verb, all sharing the same subject. An example of this taken from Glass and Bangay (2008) is shown in Fig.4.5.

The structure of these chains is captured in the dependency structure of the sentence and so by following the chain of dependents from the bottom verb, the main verb can be found and the subject can be checked. If no expressive verb was found in the previous step, this feature does not apply.

The algorithm functions by keeping a stack (a Python list can function as a stack) which originally contains the chosen expressive verb, and adding to it all verbs which govern the expressive verb. In turn, each of these verbs are popped from the stack and the verbs governing them are added to it. Every time a verb is popped from the stack, the system checks if the governor (subject) of the verb has the same position in the sentence as one of
the candidates and if it does the candidate is awarded a point. Note this algorithm is used only if the expressive verb has no explicit subject, like in the example from Fig.4.5.

Verb distance: Candidates are awarded salience based on their proximity to the expressive verb. This is performed using the same algorithm which was used for scoring verbs. If no verb was found in the previous step this feature does not apply.

4.6.4 Speaker Resolution

Regardless of whether an actor was found in the previous step, the speaker resolution process is carried out next. The function for speaker resolution works using a number of variables representing current contextual information:

- ACTOR: The actor found in the previous step.
- BEST: The character alias set from the top of the priority list of characters.
- SECOND_BEST: The character 2nd from the top in the priority list.
• PREVIOUS_SPEAKER: The identity of the previous speaker.

• PREVIOUS_QUOTE_SENTENCE: The sentence index of the previous quote (if the quote was embedded it is the index of the host sentence).

• dialogue_chain: Boolean value, true if the target quote is a stand-alone quote and the quote before it was also a stand-alone quote.

SID first checks if ACTOR is an alias of a character and if it is, this character is chosen as the speaker and the character’s alias set is returned. Similarly if ACTOR is a reference to the narrator, he/she is automatically chosen as the speaker. If the actor is anaphoric, either pronominal or deictic (no actor was found), then the system uses a series of if-else statements to try and reason as to the right speaker. A tree representation of the decision process is shown below in Fig.4.6.

Looking at the tree it is difficult to find logic in the branching pattern. This haphazard branching is a result of branches being added to solve any persistent type of error encountered during the testing of the system. Every time an error was identified which seemed to be of a certain type and predictable in terms of the context in which it occurred a branch was added to solve the problem, which in itself usually caused other errors to occur, which demanded more branches or branch restructuring. This is certainly not the best way to create a decision tree but due to time restrictions, a more logical and strategic approach was not possible. As it stands however, the current decision tree works quite well and has solved all major errors that plagued the system early in its development.

It is worth noting however that, in its current form, SID does not have the ability to logically assign a quotation no speaker. This would be desirable behaviour in the case of scare quotes. It can be seen in Fig.4.6 that some of the nodes lead to choosing the None character although this is by no means a comprehensive way of detecting scare quotes and is something that must be implemented. Furthermore, SID has no way to resolve nominal anaphora as of yet. While a noun can be chosen as the most likely speaker, SID has no way of resolving if the noun is an instance of nominal anaphora.
4.7 Sample Run-through

Next I will explain a typical run-through using SID to perform speaker attribution on a Project Gutenberg text file.

4.7.1 Setup

SID was created on a Windows system and it’s ability to run on other operating systems has not yet been fully investigated. However the only necessary changes for a different would be changing file path conventions and the Java command used to run Core tools. In order to run SID on a Windows system, Python version 3.4 along with Java is necessary, and the Python NLTK library. Using the command line, the user must navigate to the directory containing the program extract_dialogue.py. Ensure the Project Gutenberg text file to be processed is in the gutenberg_texts folder.

During processing SID will create an output folder with the name of the file to be processed with the suffix –output attached to it. If the user wants to rerun the program on the same text file, or if the process was interrupted, the user must ensure this folder and whatever contents it has are deleted before rerunning the program. The program will not run if it detects the existence of the folder.

4.7.2 Execution

In order to run the program the following command is executed via the command line:

```
python extract_dialogue.py
  gutenberg-texts\the-beast-in-the-jungle.txt
```

SID first reads the text file and checks to make sure an output directory for the file doesn’t already exist. If one does, the program exits, if not, a directory is created. SID removes the Project Gutenberg header and footer (Fig.A.2 shows a sample run through on the command line) and the dialogue is then extracted and quotes are substituted with tags. The text is then split
into sentences and printed to a file in the which is passed to the Core tools. This part of the process takes the longest, averaging around 40-50 seconds for the test text. The output is then printed to an XML file. SID iterates through the text and builds the list of characters.

At this point the user is asked to assign a gender to the characters, or to remove them. If the user makes a mistake this process can be repeated. SID then performs the speaker attribution process; verb scoring, actor scoring, and speaker resolution. The results are printed to files in the output directory. There are two output files (more files can be printed for testing if needed); one contains a list of all of the quote tags with their attributed speaker, and the other contains the set of quotations (quoted text, not tags) per character. A portion of the dialogue attributed to a character from the test text is shown below:

```json
{'Miss Bartram', 'May', 'Bartram', 'May Bartram'}
Gender: f

"No--I wasn't so free-and-easy then. But it's what strikes me now."
"Ah your not being aware of it"
"It was something about yourself that it was natural one shouldn't forget--that is if one remembered you at all. That's why I ask you,"
"not from this side. This, you see,"
"of course one's fate's coming, of course it _has_ come in its own form and its own way, all the while. Only, you know, the form and the way in your case were to have been--well, something so exceptional and, as one may say, so particularly _your_ own."
"Do you ask that, by any chance, because you feel at all that yours isn't? I mean because you have to wait so long."
"I'll watch with you,"
"More monstrous. Isn't that what you sufficiently express,"
"I met you years and years ago in Rome. I remember all about it."
"No,"
"Really."
"Well,"
```

The user is then asked if they would like to perform testing. If so the user must give the name of a gold standard test corpus (see 4.4.4 for details of
the format of a test file for SID). The file must be located in the current working directory, the same one where `extract_dialogue.py` is stored. SID then performs testing by comparing the results of SID with the gold standard results. SID tests for overall accuracy, and precision and recall per speaker in the text, and prints the results.
Figure 4.6: SID’s rule-based decision process represented as a tree. PS (previous speaker), 2ndB (second best) gm (gender match function)
Chapter 5

Evaluation and Results

In order to evaluate a speaker attribution system a gold-standard corpus is needed containing the set of quotations uttered by each character in the text. Similarly the corpora may also take the form of a list of every quotation with a label indicating the appropriate speaker. Due to the precise nature of the task, corpora such as these are not very common. Both Glass and Bangay, and Elson and McKeown created their own corpora however neither were available at the time SID was being developed. It was therefore necessary to create my own test material.

5.1 Evaluation

Ideally, to create a test corpus for a speaker attribution system one would follow the steps outlined in Elson and McKeown (2010), whereby the works of a number of different authors are annotated by more than one person, and quotes which receive unanimous agreement added to the test corpus. This was not possible for this project however so I had to settle for hand-annotating a text file to use for testing and evaluation.

The text used is *The Beast in the Jungle*, by Henry James. The text is a short story and contains 301 instances of quoted text, split between two speaking characters, and a number of unattributed scare quotes. This text was cho-
sen on the one hand for its short length, but also because Henry James has
a uniform writing style that would not pose many unforeseen problems for
SID. Furthermore, the text does not have any instances of nominal anaphora,
which my system cannot handle. I found this preferable as it means my sys-
tem is evaluated based only off of its current capabilities.

The gold standard corpus consists of an index referring to a specific quote (the
same as the \texttt{QUOTE}\_\texttt{n} tags described earlier), and the character alias set
of the speaker who utters the quote. Unattributed quotes or scare quotes are
annotated as the 'None' character. Below is an excerpt of the gold-standard
corpus:

\begin{verbatim}
<QUOTE_37>|John Marcher|John|Marcher
<QUOTE_38>|May Bartram|Bartram|May|Miss Bartram
<QUOTE_39>|John Marcher|John|Marcher
<QUOTE_40>|May Bartram|Bartram|May|Miss Bartram
<QUOTE_41>|John Marcher|John|Marcher
<QUOTE_42>|May Bartram|Bartram|May|Miss Bartram
<QUOTE_43>|May Bartram|Bartram|May|Miss Bartram
<QUOTE_44>|John Marcher|John|Marcher
<QUOTE_45>|May Bartram|Bartram|May|Miss Bartram
\end{verbatim}

5.2 Results

Tests were carried out in order to assess SID’s overall accuracy, and the
precision and recall per character. In order to ensure SID’s results were
meaningful, the results were compared to the results of a random baseline.
The random baseline assigns a random character from the text, including the
None character, to each quotation.

5.2.1 System Accuracy

The system accuracy was taken as the percentage of correctly annotated
quotations by the speaker attribution system, and the random baseline.
From the results in Tab.5.1 it is obvious that the system performs much better than that of the baseline indicating it functions as desired and its results are meaningful.

**5.2.2 Per Character Precision and Recall**

Precision and recall are measures for analysing data in terms of relevance. In terms of a speaker attribution system, the precision for a character refers to the proportion of quotations assigned to that speaker that are relevant (correctly attributed), while the recall refers to the fraction of the correct quotations for that character which are assigned to them. High precision means that an algorithm returns more desired results, than undesired results, while high recall indicates that an algorithm returns most of the set of desired results. The results for my system versus the baseline are shown in Tab.5.2. The precision and recall are much higher for my system compared to the baseline, with the exception of the *None* character.

<table>
<thead>
<tr>
<th></th>
<th>Num. Gold</th>
<th>Num. Attributed</th>
<th>Correct</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>None</td>
<td>14</td>
<td>112</td>
<td>5</td>
<td>4.46%</td>
</tr>
<tr>
<td></td>
<td>John</td>
<td>136</td>
<td>89</td>
<td>40</td>
<td>44.94%</td>
</tr>
<tr>
<td></td>
<td>Mary</td>
<td>152</td>
<td>101</td>
<td>49</td>
<td>48.51%</td>
</tr>
<tr>
<td>SID</td>
<td>None</td>
<td>14</td>
<td>0</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>John</td>
<td>136</td>
<td>144</td>
<td>129</td>
<td>89.58%</td>
</tr>
<tr>
<td></td>
<td>Mary</td>
<td>152</td>
<td>158</td>
<td>146</td>
<td>92.41%</td>
</tr>
</tbody>
</table>

Table 5.2: Precision and recall per character data for baseline and SID.

Again it is clear that SID performs well over the baseline. The per character data also highlights SID’s inability to properly identify scare quotes as having no speaker (indicated by attributing 0 quotations to *None*). For the majority of cases SID will always choose a named character as a speaker. While this is a problem, I still find the results for the speaking characters quite promising.
5.3 Analysis

Due to limited testing performed on the system it is not possible to make concrete statements about SID’s performance. Nevertheless, looking purely at the data analysed, the following are some observations about the current state of the system.

- The system performs very well achieving overall accuracy of around 91.06%.
- Precision and recall scores are also very high per character.
- SID can successfully extract all quotations, but does not filter out scare quotes nor can it assign them the None character.
- SID system has functionality for detecting a narrator and treating them as a character (tests were carried out on different texts to show this).
- The system can resolve pronominal and deictic anaphora but it cannot resolve nominal anaphora.
Chapter 6

Conclusion

The aim of this project was to create a speaker attribution system that could extract quotations from narrative texts and attribute a speaking character to each quotation. SID performs quite well, achieving overall accuracy of approximately 91.06%, although because of the limited testing carried out, the results are only indicative of its performance for the specific test material. Regardless, the results show that it is possible to create a nave speaker attribution system which does not require complex statistical learning techniques. SID is quite a simple system, and I think it functions in a way which is in line with the cognitive faculties humans use when performing the task themselves. SID is also flexible enough in its structure that further work should be possible to improve performance, but also add functionality that wasn’t implemented due to time constraints. Next I will discuss further work that may be added with future iterations of SID.

6.1 Future Work

6.1.1 Corpus Building

One of the major factors working against me whilst developing SID was the lack of a large, rich corpus consisting of different texts from a variety of authors. This greatly hindered the testing which could be performed, meaning
it was difficult to analyse the system’s errors and come up with appropriate solutions. It would be my intention therefore, given more time, to create a speaker attribution corpus which I could use to perform comprehensive testing. This corpus could also be made freely available for other developers to use which would make it easier for research in this area to be carried out.

6.1.2 Further Testing and Error Analysis

With a full corpus further testing could be carried out which would enable a comprehensive analysis of the errors made by the system. Using only the one text for testing meant that SID’s current decision tree in is likely subject to over-fitting, meaning SID works very well for that text, but only that text. As mentioned in 4.6.4, the branching order in the current tree was chosen solely on the test document. This means that every branching point was made based on the presence of an error that may in fact not be prevalent across all texts. By making the tree fit to these possible error outliers, its potential to work well on different texts is reduced.

With the ability to detect persistent error types across many texts the decision tree could be constructed in a much more logical, and therefore more compact and efficient way. Looking at the existing tree (Fig.4.6) it is easy to identify large parts of the tree which contain a lot of repetition. Restructuring the tree would help solve some of these problems but developing the tree from scratch would be the ideal solution.

While I was against using statistical learning methods in this iteration of the system, I can see how having some advanced methods could help in constructing a better decision tree. Because the decisions made by the tree in most cases rely solely on the variables described in 4.6.4, using decision tree learning algorithms and an annotated corpus would allow us to figure out which of these variables is most crucial in deciding on the speaker. Knowing how important a variable is in deciding a speaker means a decision tree could be constructed which uses the variables in the order which will resolve a speaker the fastest. Performing this type of statistical learning however would require a speaker attribution corpus where every quotation is labelled with the values of these variables used when resolving the speaker.
6.1.3 Parameter and Property Settings

Creating a speaker attribution that caters to every style of narrative text is something that has yet to be achieved, which isn’t surprising considering how wildly different narrative texts can be. However it is possible to create systems which can be configured to suit the type of text being processed. In regards speaker attribution, the ability to configure the properties of SID to suit the style of the text would be advantageous. For example some type of texts may tend to use more direct speaker references, or some form of anaphora more than others. Depending on the type of text, the features and variables needed for the speaker attribution process may have very different priorities. Therefore being able to alter the importance of such parameters would mean SID would be able to adapt to the type of text being processed, thus broadening its scope.

6.1.4 Nominal Anaphora Resolution

SID can currently solve pronominal and deictic anaphora however it is not clear to what extent it can solve nominal anaphora. While my system has the ability to choose a common noun as a speaker, it does not identify nouns as being possible coreferents of a speaker antecedent. To add this functionality a method for identifying character nominals is necessary as well as a comprehensive coreference resolution system.

Stanford Core NLP includes a coreference resolution tool however I found using it resulted in my system taking considerably longer to run (many minutes more). This is due to the fact that the tool, by default, performs coreference resolution for every entity it detects, not just people. The tool’s properties can be configured to filter out certain types of entities however this has yet to be tested. If setting the properties means improved performance, the Stanford coreference tool would be very beneficial in making nominal anaphora resolution possible, which would also eliminate the need to develop a system from scratch.
6.1.5 Other Languages

I am not aware of a speaker attribution system in any other languages apart from English meaning it would be of great interest to me to adapt SID to work for another language, most likely German which I am studying as part of my degree. Theoretically, most of SID’s methodology for performing the speaker attribution task should apply directly to German. However there are a few language specific problems.

Firstly, in order to extract dialogue a different regular expression is necessary as quotations in German texts are usually marked with opening << and closing >>, rather than double or single quotation marks.

German capitalises all nouns meaning the character identification task would need to be drastically altered to filter out all common noun mentions. German also has three grammatical genders, meaning when a common noun antecedent is referenced using a pronoun, the pronoun relates to the noun’s gender. What this means is that extracting possible actor candidates is made more difficult as most instances of the English ‘it’ will be replaced with the German version of ‘he’ or ‘she’. The three line gloss of a German-English translation below illustrates this:

(1) Die Frau sprach mit dem Hund und er bellte sie an.
(2) The woman spoke with the dog and he barked she at.
(3) The woman talked to the dog and it barked at her.

It would be very easy for SID to choose ‘er’ as the actor, as ‘er’ can refer to proper nouns but also common nouns. This means the coreference resolution system would need to be much more comprehensive for German in order to ascertain if a pronoun like ‘er’ is referring to a man, or just a male object, like a chair.

6.2 Concluding Remarks

Looking at SID in its current form, I am proud of what I have especially as it seems to work quite well. I had hoped that SID would end up functioning in
a new or unique way, however his methodology was heavily inspired by the
system in Glass and Bangay (2008) meaning SID isn’t very original. However
considering the small amount of work in the area of speaker attribution, there
is now ample opportunity to build on the foundation that SID provides and
look at the problem in new and innovative ways. In the future I hope to build
on SID, with the hope of developing functionality that will give it a greater
understanding of the narrative flow of the text and to emulate as much as
possible, the human process for speaker attribution, starting with the ability
to glean information from within quotes themselves.
References


International Symposium of the Pattern Recognition Association of South Africa, 1-6.


Appendix
Figure A.1: The WordNet website showing the hypernyms of the word 'shout'. The hypernyms indicate that ‘shout’ is a verbally expressive word.
C:\Users\Stephen\Documents\college\fyp\novel_parser>python extract-dialogue.py gutenberg-texts\the-beast-in-the-jungle.txt

-------------------------------
| QUOTATION PARSER |

Removing header and footer...
Done.

Extracting dialogue...
Done.

Running stanford NLP tools...

Adding annotator tokenize
annotator.tokenize: No tokenizer type provided. Defaulting to PTBTokenizer.
Adding annotator coref
Adding annotator pos
Reading POS tagger model from edu/stanford/nlp/models/pos-tagger/english-left3words-distsim.tagger ... done (2.4 sec).
Adding annotator lemma
Adding annotator ner
Loading classifier from edu/stanford/nlp/models/ner/english.all.3class.distsim.crf.ser.gz ... done (7.0 sec).
Loading classifier from edu/stanford/nlp/models/ner/english.muc.7class.distsim.crf.ser.gz ... done (16.5 sec).
Loading classifier from edu/stanford/nlp/models/ner/english.conll.4class.distsim.crf.ser.gz ... done (18.3 sec).

Initializing JollyDayHoliday for sutime with classpath:edu\stanford\nlp\models\sutime\jollyday\Holidays_sutime.xml
Reading Tokens/Regex rules from edu\stanford\nlp\models\sutime\default.sutime.txt
Reading Tokens/Regex rules from edu\stanford\nlp\models\sutime\english\default\sutime.txt

06. 28\th 15:04:15 PM edu.stanford.nlp.ling.tokensregex.CoreMapExpressionExtractor.appendRules
INFO: Ignoring inactive rule: null
06. 28\th 15:04:15 PM edu.stanford.nlp.ling.tokensregex.CoreMapExpressionExtractor.appendRules
INFO: Ignoring inactive rule: temporal-composite-\ranges
Reading Tokens/Regex rules from edu\stanford\nlp\models\sutime\english\holidays\sutime.txt

Adding annotator depparse
Loading depparse model file: edu\stanford\nlp\models\parser\ndep\english\SD.ser.gz

PreComputed 000000, Elapsed Time: 2.95 (s)
Initializing dependency parser done (19.8 sec).

Ready to process: 1 files, skipped 0, total 1
Processing file C:\Users\Stephen\Documents\college\fyp\novel_parser\the-beast-in-the-jungle-input\stanford-input-the-beast-in-the-jungle.txt ... writing to C:\Users\Stephen\Documents\college\fyp\novel_parser\the-beast-in-the-jungle-output\stanford-input-the-beast-in-the-jungle.txt (35.613 seconds)
Annotating file C:\Users\Stephen\Documents\college\fyp\novel_parcer\the-beast-in-the-jungle-output\stanford-input-the-beast-in-the-jungle.txt (38.917 seconds)
> (38.917 seconds)
Processed 1 documents
Skipped 0 documents, error annotating 0 documents
Compilation pipeline timing information:
  TokenizerAnnotator: 0.2 sec.
  WordToSentenceAnnotator: 0.1 sec.
  POSTaggerAnnotator: 1.4 sec.
  MorphologyAnnotator: 0.2 sec.
  NERCombinerAnnotator: 26.1 sec.
  DependencyParserAnnotator: 7.1 sec.
  TOTAL: 35.6 sec. for 1976 tokens at 555.1 tokens/sec.
  Pipeline setup: 0.0 sec.
  Total time for StanfordCoreNLP pipeline: 39.0 sec.
---
Done.

Building speaker candidate list...
Done.

In order to ensure proper genders are assigned to the speakers you will be asked to choose a gender for each avatar set, or remove the speaker if it has been incorrectly extracted.

If there is a narrator in the text label it as 'n'. It will appear as
<<NARRATOR>>. If there is no narrator <<NARRATOR>> should be removed.

n : male, f: female, ni: narrator, r: remove speaker

('David Price', 'Price', 'David')
...'
('John Marcher', 'Marcher', 'John')
...'
('Mary')
('Martin', 'Martin Secker', 'Secker')
...'
('Soronto')
...'
('NARRATOR')
('Miss Bartran', 'May', 'Bartran', 'May Bartran')
...'
To be removed: ('David Price', 'Price', 'David')
Gender for ('John Marcher', 'Marcher', 'John') is 'm'
To be removed: ('Mary')
To be removed: ('Martin', 'Martin Secker', 'Secker')
To be removed: ('Soronto')
To be removed: ('NARRATOR')
Gender for ('Miss Bartran', 'May', 'Bartran', 'May Bartran') is 'f'

If you agree with these choices, press 'y', otherwise 'n'.
...y
If you agree with these choices, press 'y', otherwise 'n'.

...y

Candidate speakers and their genders:

FEMALE: ('Miss Bartram', 'May', 'Bartram', 'May Bartram')

MALE: ('John Marcher', 'Marcher', 'John')

Reading text and resolving speakers to each quotation...

Done.

Printing output...

Done.

Perform testing? y/n

...y

Please enter name of gold standard test file.

...the-beast-in-the-jungle-test.txt

Number of correctly annotated quotations by BASELINE:

100

Percentage of correctly annotated quotations by BASELINE:

35.762%

Number of correctly annotated quotations by SYSTEM:

225

Percentage of correctly annotated quotations by SYSTEM:

81.382%

Per character accuracy by BASELINE...

Percentage of correctly annotated quotations for ('None')

42.862%

Percentage of correctly annotated quotations for ('John Marcher', 'Marcher', 'John')

59.712%

Percentage of correctly annotated quotations for ('Miss Bartram', 'May', 'Bartram')

31.582%

Per character accuracy by SYSTEM...

Percentage of correctly annotated quotations for ('None')

4.42%

Percentage of correctly annotated quotations for ('John Marcher', 'Marcher', 'John')

94.052%

Percentage of correctly annotated quotations for ('Miss Bartram', 'May', 'Bartram')

96.722%

Precision and recall per speaker by BASELINE...

('None')
<table>
<thead>
<tr>
<th>Speaker</th>
<th>Gold</th>
<th>Found</th>
<th>Correct</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>14</td>
<td>101</td>
<td>6</td>
<td>5.9%</td>
<td>42.8%</td>
</tr>
<tr>
<td>John Marcher</td>
<td>136</td>
<td>104</td>
<td>54</td>
<td>51.9%</td>
<td>39.2%</td>
</tr>
<tr>
<td>Miss Bartram</td>
<td>152</td>
<td>97</td>
<td>48</td>
<td>49.4%</td>
<td>31.5%</td>
</tr>
</tbody>
</table>

Figure A.2: (1), (2), (3), (4) show the console interface as a file is processed by SID.