# PlayVis:
# Analyzing and Visualizing Character Interactions in Plays

**Mike Nolan**
Rochester Institute of Technology
Rochester, NY 14623
mpn3712@rit.edu

**Tyler Krupicka**
Rochester Institute of Technology
Rochester, NY 14623
ttk2229@rit.edu

## Abstract

The intent of this project is to create a platform for analyzing and visualizing key events and character interactions throughout plays. Our goal is to be able to meaningfully extract important data about characters; including their names, aliases, lines, interactions, and sentiment towards each other. In this document we present the results of our research automatically locating characters, performing sentiment analysis, and organizing data in to an object that can be used for visualizations. We also discuss how this technique can be used for analyzing other plays.

## 1 Introduction

In the context of plays and written literature, a combination of text and sentiment analysis provides an opportunity to better display character information. Over the course of plays, it becomes difficult for a reader to keep track of every interaction between characters. By classifying interactions as events over time, however, it is possible to keep track of this data and format it in a way that makes it accessible at a glance. By experimenting with ways for extracting and displaying this data, we researched and demonstrated event tagging and sentiment analysis in a form that makes it useful for surveying the data.

This research aimed to create a system to locate characters and generate useful information from the online Gutenberg library of plays. However, there are a few unique challenges to overcome when organizing information from this data set. Character names in each play do not remain consistent, which requires extra analysis to distinguish characters from each other. Once we have determined names, we also had to locate lines and sets of lines to attribute to the character; while at the same time recording the expressed sentiment and person being referred to.

This research has a few different applications for processing data. Our program and implementation can be used to locate character names and lines in many plays that follow the standard line format. The program also can find approximate sentiment in each of the lines and direct the sentiment to another character; while also creating a JSON object which can be used in outside interpretations and visualizations. The intent of this research was to create a platform for further research on character analysis in literature.

## 2 Previous Work

Previous work has been done to generate sentiment analysis and generalize character to character relations in plays (Nalisnick and Baird, 2013). Researchers at Lehigh used the same valence value sentiment analysis technique to compare sentiment between characters in Shakespeare's plays, and were able to get good results. Other research has focused specifically on finding more accurate sentiment analysis on diverse phrases (Lokciyan, Celebi, Ozgur, and Uskudarli, 2013).

This research aims to expand on this external research by providing more generalized character extraction and organization, as well as providing further relevant information beyond sentiment and finding ways to export it.

## 3 Creating Character Classes

In order to extract any useful information about how characters interact over the course of a given play, we needed to first identify character names and associate their lines with them. The Project Gutenberg translations often use inconsistent spacing and character names, so extra analysis is required to determine true names

throughout the text. The goal was to create class objects that contained each character's full name, which could be used to associate aliases and lines with.

Finding characters would be useful in context of each line as well. By parsing sentences in to a recognizable grammatical form, it should be possible to predict whether or not a word is a name based on its location in the sentence (Cranenburgh, 2012).

### 1.1 Name Recognition

To find full character names, the program first identified where the play introduces each character. In normal play dialogue, the document consistently uses aliases for each character (for example Hamlet is often abbreviated to "Ham") so we could not assume that each line contains the full name.

After looking at the names, aliases, and repeated formatting through the document, we decided to use regular expressions to initially locate the name/alias assigned to each line. Each character line begins with white space, followed by the name/alias and then some punctuation (usually a period). Our regular expression gave us the name assigned to the line which we then used to compile a preliminary list of names and aliases.

**Example:**

*" Ham. Upon the talk of the poisoning?"*

"Ham" is stored as a possible alias.

The study also made use of the entrance lines in the play to extract full length character names. Each of these lines begins with the words "Enter" or "Re-enter," which made initially locating them easy, however they often contain information besides just names. For each token in the line, we checked if the word is in a dictionary of known English words, using the pyEnchant dictionary plugin with python. If the token is not determined to be a word, or contains an upper case first letter, we can then assume that it is a character name.

**Example:**

*"Enter Horatio and Marcellus."*

"Horatio" and "Marcellus" are stored as names.

The program compiles a comprehensive list of possible character names and aliases, and then compares each of the elements in the list to determine aliases or abbreviations. If an element in the list was contained at the beginning of a longer name in the list, we assumed that it must have been an alias for the longer name.

### 1.2 Recognizing Lines

Once the software had a list of character names and their aliases we can traverse the document and record the lines attributed to each character. As it located a line that begins with a given name or alias, it recorded the line as a 'start' for that character and then continued reading lines until it hit a break. The last line recorded becomes the 'end' line. In this way we were able to assign each block of text as an individual line for the appropriate character.

This model ended lines based on two conditions: either another character line begins, or there is a break in text. The start and end value can then be used later on to locate the entire block of speech for sentiment analysis.

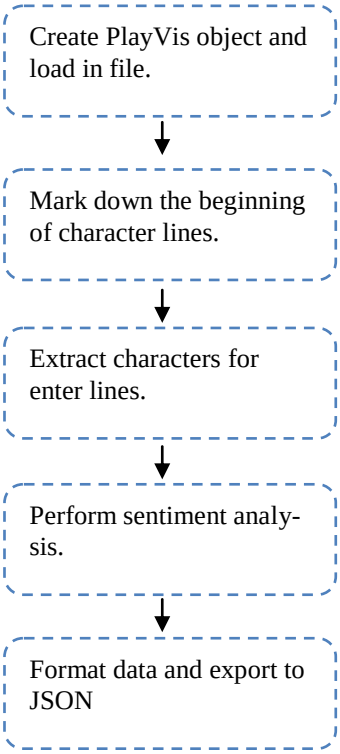


Figure 1: Progression of data

### 1.3 Last Character Recognition

In order to find any form of sentiment in character lines we had to determine who the line's content was directed at. We decided that the most reliable solution would be to assume that each line was directed at the person who spoke immediately before. This works in many back-and-forth conversations as well as situations where

there are many different people in the scene, but is inaccurate in lengthy speeches or instances where characters are talking about someone else. Despite this, the system generally produced enough direction to create a model off of. This is an area where it would be advantageous to do further study, as it is difficult to find who the speech is directed at without gathering data from the speech itself.

As each of the lines was assigned to a character, the program kept track of the last character that spoke and then saved their name along with the line that refers to them. For example, if Hamlet is in a discussion with Horatio, we save that his lines during that conversation are directed at Horatio.

## 4 Line-by-Line Sentiment Analysis

This study used fixed valiance value sentiment analysis to provide a sentiment value for each of our designated lines. This solution utilized the Lexicoder Sentiment Dictionary (LSD) to create a dictionary with sentiment values for keywords. Each word in a set of lines is checked in the sentiment dictionary, and adjusts the line's overall sentiment if the word has a positive or negative attribute. For example: a line containing only the word 'hate' will have a sentiment value of -1, while a line only containing 'love' will have +1. A line containing both 'love' and 'hate' would therefore have a neutral sentiment of zero.

Sentiment values are stored along with the line location and last character. The difficulty with using in-line sentiment analysis is that more often than not a line will not contain any of the sentiment keywords. Additionally, older plays contain a different vocabulary from what modern dictionaries will account for, making it less likely to find keywords. This results in many lines being classified as neutral, which has less overall meaning for the text. The main way to combat this would be to find more comprehensive sentiment tagged dictionaries.

Other researchers have found ways to include more in-context based sentiment analysis using classifiers. Systems for generating linguistically motivated features and finding context for lines in outside sources increases allow for much better context for sentiment analysis (Csomai and Mihalcea, 2008). Sentiment analysis through web mining and part-of-speech based models also makes predicting sentiment much more accurate (Wu and Wen, 2010). Finding context in the data is going to be more accurate and adaptable depending on the method.

## 5 JSON and Visualization

To allow our data collected in our python program to be used and interpreted by our JavaScript visualization engine; we needed to provide output to a JSON file which consisted of time stamped data of characters lines, sentiment, and which character this line is directed at. Due to the formatting of our character data in our python application, our algorithm to format the data in a time stamped fashion was sub-optimal but on both of our machines we never saw a decrease in performance.

The JSON file timestamps data by ordering each of our blocks of speech chronologically in an array. Our visualization engine would then load in the JSON file and loop over the data in a set period of time, slowly filtering in new data from the JSON file. This allows the user to see how sentiment towards characters changes over the course of the play.

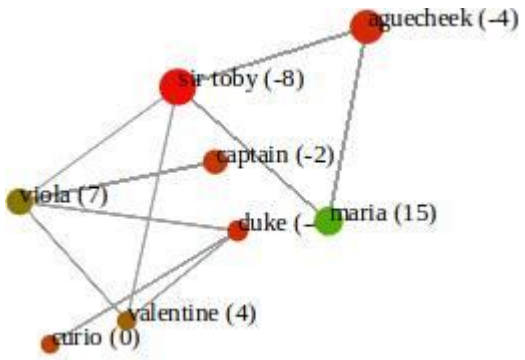


Figure 2: Node based visualization

Visualizing the data was a challenge for us. Given the large amount of data that we were interpreting, the task of showing that data in a manner that made it easy to consume required lots of research into the subject of info vis. We looked into the use of multiple bar graphs, tables, and node graphs and decided on node graphs. Not only did the use of node graphs create a visually interesting aesthetic but also combined with the use of color, allowed us to easily convey sentiment between characters. It also allowed us to integrate animation.

Instead of writing our own visualization library from the ground up, we decided to use d3.js. D3 proved to be efficient while also allowing a large amount of customization when creating the visualizations. The library did add complications into

the process of creating visualizations. Complications arose due to the library's large scope and minimal documentation. With time and some trial and error we were able to get a prototype up and running.

## 6 Results

There were multiple outcomes for this research due to the multiple techniques of character extraction and sentiment analysis that was implemented in the study.

### 6.1 Exporting Classes and Visualizing

| Text | # of Characters |
|---|---|
| Hamlet | 85 |
| Midsummer Night's Dream | 65 |
| Julius Caesar | 80 |
| MacBeth | 69 |
| Romeo and Juliet | 100 |

Table 3: Recognized characters per play

When visualizing our data we found that plays with much smaller amount of characters often times created a much more comprehensive graph. In future work it might be useful to remove nodes that aren't relevant to the current "conversation". This could be done by taking the lower percentile of nodes and after a certain time of inactivity removing them from the graph. It is also possible to retrieve the main characters from the dramatis personae.

### 6.2 Sentiment Analysis

Our method of using fixed valiance value sentiment analysis gave inconsistent results across the variety of plays that we analyzed. We suspect the cause of this is our limited lookup table for sentiment values. We also noticed that there were many cases in which sentiment could be misinterpreted.

| Title | Lines w/o Sentiment | % Error |
|---|---|---|
| Hamlet | 1008/2244 | 44% |
| Midsummer Night's Dream | 416/1008 | 41% |
| Julius Caesar | 658/1586 | 41% |
| MacBeth | 286/396 | 72% |
| *Romeo and Juliet* | *228/1642* | *38%* |

Table 4: Sentiment Retrieval Rate

We expect that our sentiment analysis could be improved by integrating systems to perform named entity recognition for each of the characters blocks of text. This would allow us to do a better job of recognizing subjects that characters were speaking about and then attributing the sentiment towards that.

| Study | Correct Lines | % Correct |
|---|---|---|
| All annotated lines | 36/50 | 70% |
| Mislabeled sentiment in incorrect lines | 2/14 | 87% |

Table 5: Human sentiment review of Hamlet

In order to better understand the accuracy of the sentiment dictionary we reviewed a set of fifty lines from Hamlet. Lines were labeled as incorrect when they were different from the annotator (lines were labeled positive, negative or neutral.) Alternatively, lines were recorded as "mislabeled" when they had opposite sentiment – positive vs. negative or vice versa. This small sample showed that despite the inherent flaws of a dictionary model, only two of the total lines were misinterpreted. The twelve other incorrect lines reflect sentiment that was not assigned, which was negligible in the scope of this research.

When annotating, we also noted that the use of early modern English in the Shakespearian plays made sentiment attribution difficult for human review as well. This is an important consideration when extracting sentiment on plays from different time periods.

An element of sentiment analysis that would help determine sentiment over a greater variety of texts is polarity shifting (Sentiment Classification, 2010) Sentiment generally shifts over the course of written literature, which means that if machine learning could detect different changes it could predict sentiment between gaps in direct statements.

### 6.3 Exporting Classes and Visualizing

The application managed to collect and output a list of events for each play that was given as input. With further investigation we managed to find some discrepancies in our data set. After

some tuning of the application, the output was consistently feeding lists of events that lined up with the given play. All graphics were generated with D3.js JavaScript library. This visualization includes features such as the interactions themselves, prominence of each character, sentiment received by other characters, and some meta-data on each character such as their name.

## 7 Conclusion

This application was able to collect and record information on how characters connected and conversed with each other. It also analyzed sentiment within the conversations themselves and output that data in a chronological, event based format. With that new data set, the visualization engine was able to create meaningful visualizations to show the end user how characters interacted with each other over the course of the play.

The model performed well at retrieving and identifying characters but due to sparse data in our sentiment dictionaries, our application accuracy of limited. For future improvement we plan to analyze context around sentiment heavy words to remove any false-positive cases. With our framework we hope to help create possibilities of quickly and easily visualizing texts by separating characters into entities and showing connections between them.

## 8 Limitations

The complexity of sentiment analysis proved to be a large challenge. When performing a fixed valiance value sentiment analysis method we found it difficult to find large manually labeled sentiment dictionaries to obtain accurate and consistent sentiment classification.

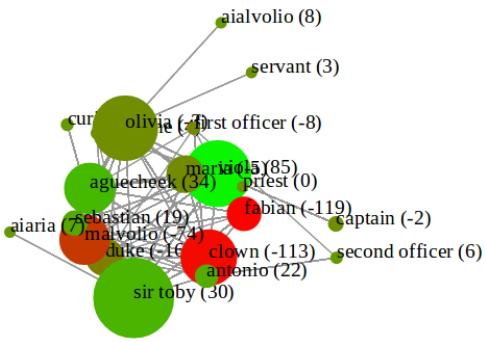


Figure 6: Crowded visualization

When visualizing plays, we often found that plays had large amounts of characters all interacting with each other at one time. This would lead to graphs that could often look crowded and that are difficult to read.

Using sentiment dictionaries also ended up being troublesome when working with texts from different time periods. Many insults and other phrases that could describe sentiment are specific to their time period and when working with text from early modern English all the way present day we had a very large vocabulary to cover.

Our largest initial problem was making our application capable of interpreting any play from the Gutenburg Project. While the Gutenburg Project provided a style guide for those who wished to transcribe books, we still found many cases where formatting was not consistent. Our use of regular expressions greatly reduced those occurrences and sometimes created characters out of objects and narrative elements.

## 9 Further Research

This research provides opportunities for further study of character extraction, sentiment analysis, and data visualization in plays and other digitized literature. A majority of the application would be improved with the implementation of name entity recognition of other ways of extracting information from lines in the play. The current method does not account for names or sentiment in context, which allows for greater inaccuracy.

Further study is also required in processing text from different time periods. This research is one of many different applications for literature that does not use entirely modern English; all of which would benefit from further study of sentiment, meaning, and vocabulary available in older literature.

## 10 Division of Labor

**Michael Nolan:**

- Co-authored code for identifying character names based on line beginnings, and set up git repository.
- Wrote base original classes and code for locating character lines.
- Created PlayVis class and all necessary formatting and storage for information.
- Responsible for generating the JSON object and creating data visualization in Javascript using D3.js.
- Assisted with research paper and presentation.

**Tyler Krupicka:**

- Co-authored code for identifying character names based on line beginnings.
- Wrote code for identifying character entrances.
- Wrote code for creating sentiment dictionaries and generating sentiment analysis values.
- Wrote a majority of the research paper, organized information.

## Software Use

- Python 2.7.6
- pyEnchant dictionary
- D3 JavaScript library

## Acknowledgments

This document was created with advice and mentorship from Professor Ovesdotter Alm at Rochester Institute of Technology. We would also like to thank Mike Bostock who created the d3.js visualization library that all of our visualizations were based off of.

## References

Eric T. Nalisnick and Henry S. Baird. 2013. *Character-to-Character Sentiment Analysis in Shakespeare's Plays.*

Andras Csomai and Rada Mihalcea. 2008. *Linguistically Motivated Features for Enhanced Back-of-the-Book Indexing.*

Yunfang Wu and Miaomiao Wen. 2010. *Disambiguating Dynamic Sentiment Ambiguous Adjectives.*

Andreas van Cranenburgh . 2012. *Literary authorship attribution with phrase-structure fragments.* Royal Netherlands Academy of Arts and Sciences.

Shoushan Li, Sophia Yat Mei Lee, Ying Chen, Chu-Ren Huang , and Guodong Zhou. 2010. *Sentiment Classification and Polarity Shifting.*