Text Networks

Text as Data Course (https://cbail.github.io/textasdata/Text_as_Data.html)
Chris Bail, PhD
Duke University
www.chrisbail.net (http://www.chrisbail.net)
github.com/cbail (https://github.com/cbail)
twitter.com/chris_bail (https://twitter.com/chris_bail)

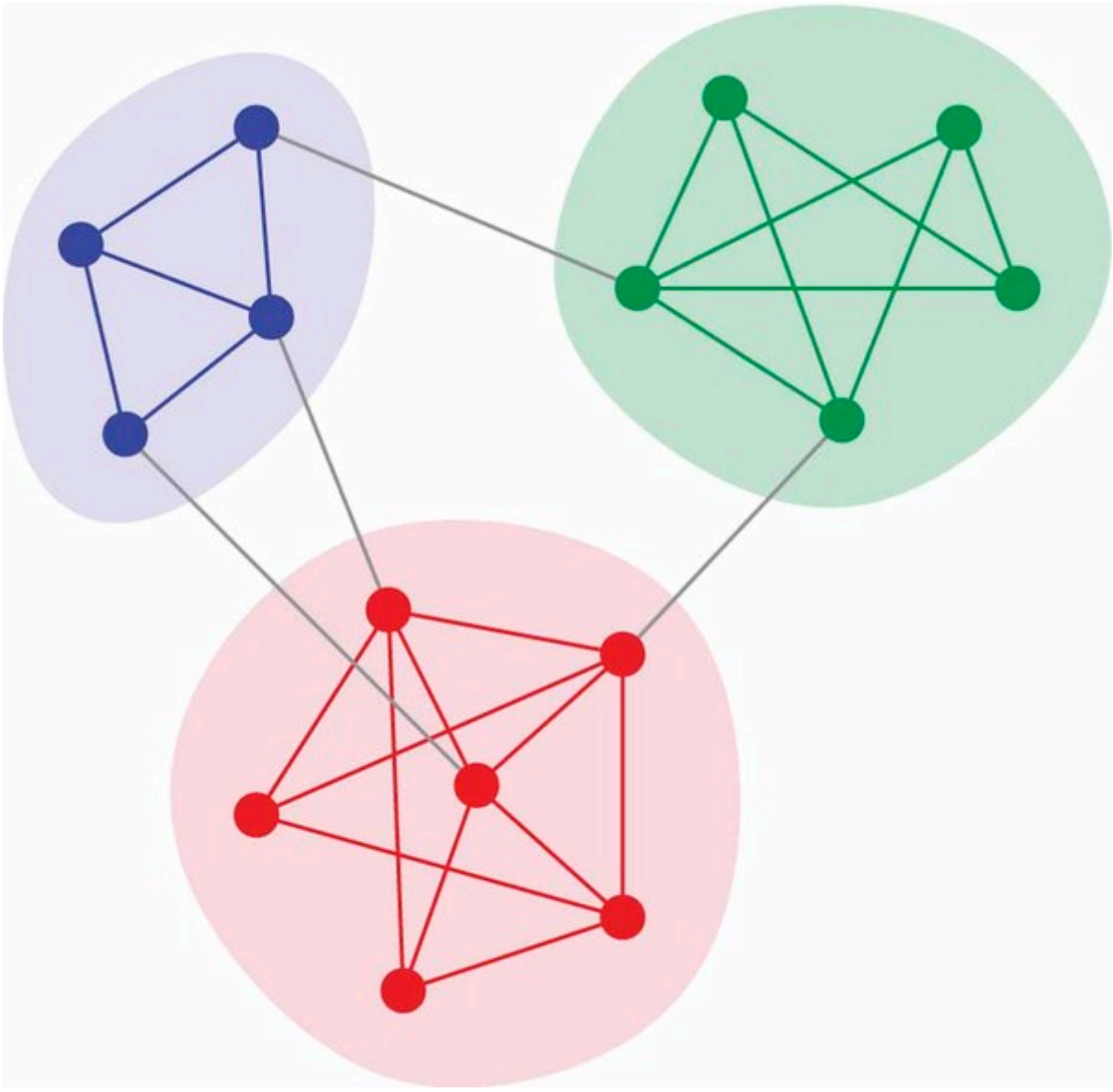
Introduction

This is the last in a series of tutorials designed to introduce quantitative text analysis in R. This tutorial focuses upon one of the newest methods in this field, called Text Networks.

What is a Text Network?

Network analysis refers to a family of methods that describe relationships between units of analysis. A network is comprised of nodes as well as the edges or connections between them. In a social network—such as the one in the figure below—nodes are often individual people, and edges describe friendships, affiliations, or other types of social relationships. A rich theoretical tradition in the social sciences describes how patterns of clustering within

social networks—and an individual's position within or between clusters— is associated with a remarkably wide range of outcomes including health, employment, education, and many others.



Though network analysis is most often used to describe relationships between people, some of the early pioneers of network analysis realized that it could also be applied to represent relationships between words. For example, one can represent a corpus of documents as a network where each node is a document, and the thickness or strength of the edges between them describes similarities between the words used in any two documents. Or, one can create a textnetwork where individual words are the nodes, and the edges between them describe the regularity with which they co-occur in documents.

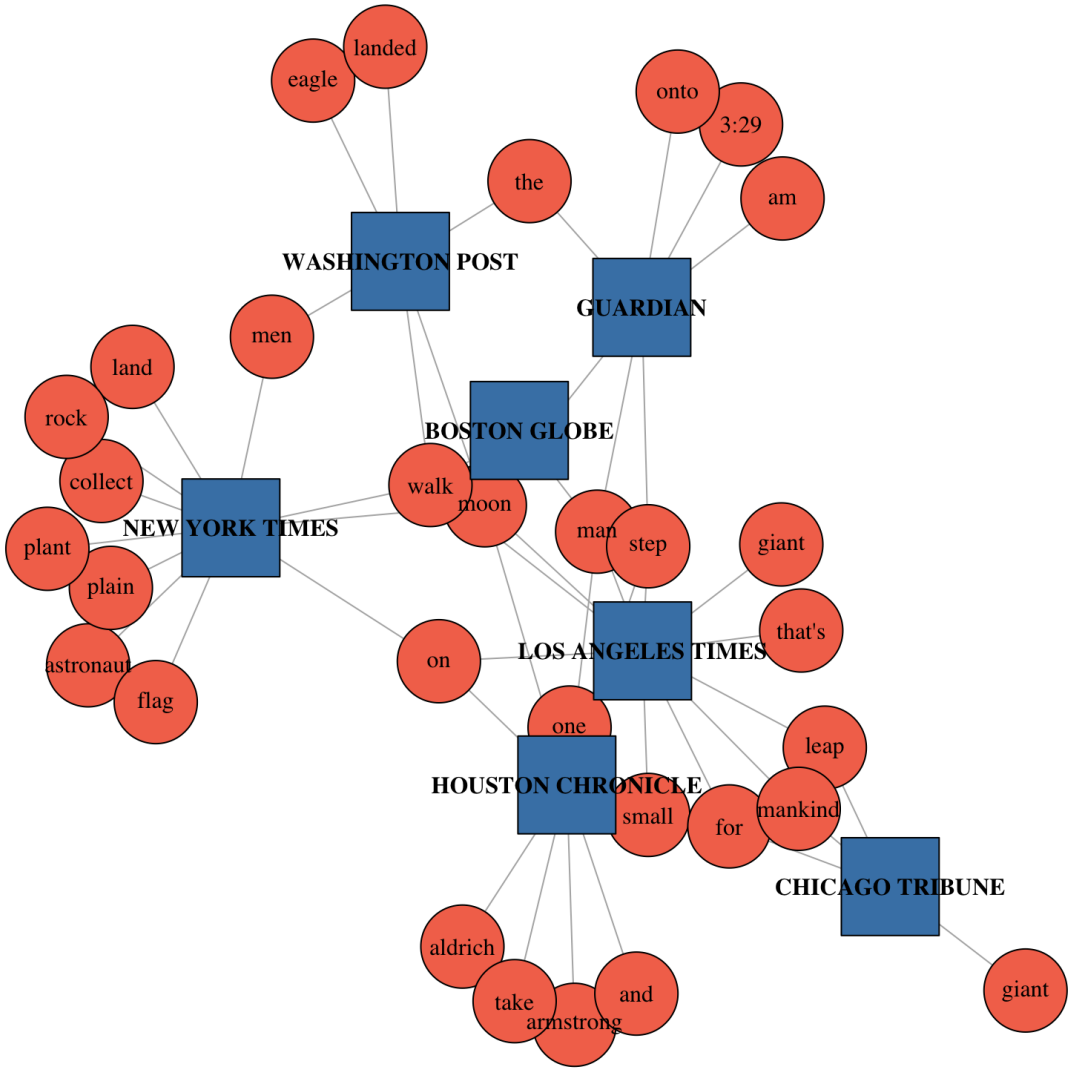
There are multiple advantages to a network-based approach to automated text analysis. Just as clusters of social connections can help explain a range of outcomes, understanding patterns of connections between words helps identify their meaning in a more precise manner than the "bag of words" approaches discussed in earlier tutorials. Second, text networks can be built out of documents of any length, whereas topic models function poorly on short texts such as social media messages. Finally, there are an arguably more sophisticated set of techniques for identifying clusters within social networks than those employed in other automated text analysis techniques described in my earlier tutorials.

Two-mode networks

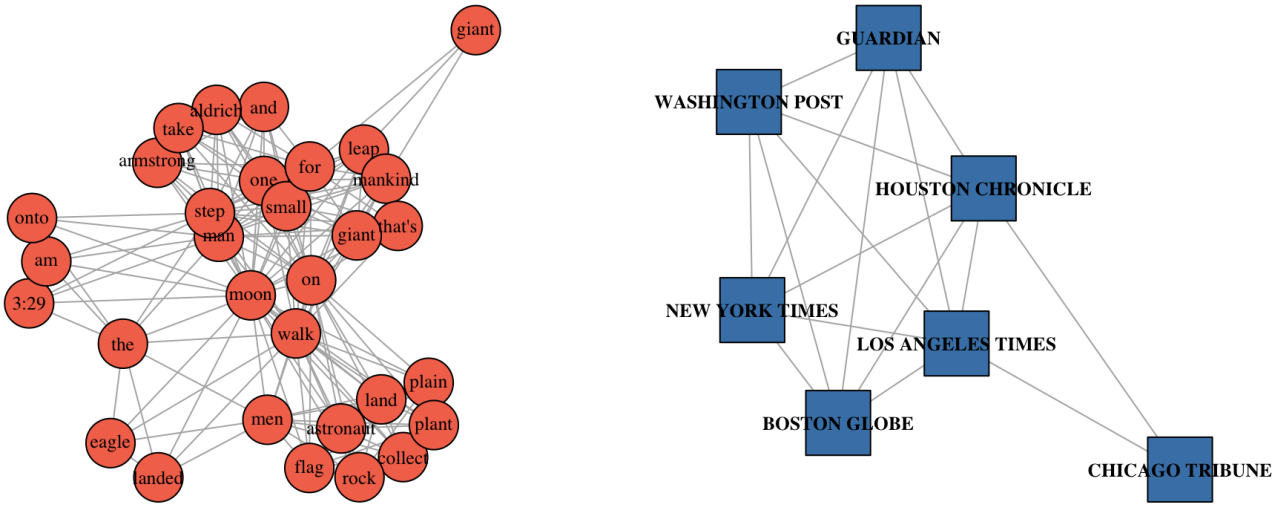
Before we move on to a working example, we will need to delve a little bit deeper into some terminology from network analysis—specifically, the concept of two-mode networks. To clarify this concept, let's take the example of a network where the first node set is words found in US newspaper headlines on the day of the first moon landing (July 20, 1969), and the second node set is the newspapers themselves. The data would look something like this:

groupvar	textvar
The Guardian	3:29 am Man Steps Onto the Moon
New York Times	Men Walk on Moon - Austronauts Land on Plain, Collect Rocks, Plant Flag
Boston Globe	Man Walks on Moon
Houston Chronicle	Armstrong and Aldrich Take One Small Step for Man on the Moon
Washington Post	The Eagle Has Landed Two Men Walk on the Moon
Chicago Tribune	Giant Leap for Mankind
Los Angeles Times	Walk on Moon That's One Small Step for Man, One Giant Leap for Mankind

Here is a two-mode projection of this network. As you can see, edges are only drawn between newspapers and words (i.e. nodes belonging to different sets).



With some reshaping of the data, this two-mode network can be projected in either of its one-mode forms. That is, one can either create a network where newspapers are connected by their use of the same words, OR, words in all of the articles can be connected based upon their co-appearance in newspapers.



In text networks, one node set will always be comprised of the words found in the documents analyzed; the other node set can be the documents themselves (as above), or some other type of meta data about those documents (such as the author's name or the date when the document was published or created).

For a more detailed introduction to text networks see Bail (2016) or Rule et. al (2015). The references to both of these articles are below.

The `Textnets` Package

The only R package presently available to implement text network techniques is the `textnets` package. The most current version of the `textnets` package is currently available on Github. To install `textnets` —or any other package hosted on Github— you will need the `devtools` package (see code below). `textnets` can take some time to install because it has a number of dependencies that are themselves complex packages.

```
library(devtools)
install_github("cbail/textnets")
```

An overview of `textnets`

The `textnets` package provides the following functions:

1. preparing texts for network analysis
2. creating text networks
3. visualizing text networks

4. detecting themes or “topics” within text networks

We will work through each of these steps one-by-one with a working example in the following sections.

Preparing Texts

The `textnets` package requires text that is contained within a dataframe, where each row represents a document. The text of each document must be contained within a single column, but the dataframe can also include other columns that describe meta data such as the author’s name or date of publication.

Example: State of the Union Addresses

To get a better sense of this, let’s take a look at some sample data. We are going to create a text network using texts from the State of the Union Addresses by U.S. presidents. Each row in the dataframe we load below describes an address given by a president, and the year in which that address was made. The dataset also describes the president’s party affiliation. This dataset is stored within `textnets` and we can load it and browse it as follows:

```
library(textnets)
data("sotu")
```

The `textnets` package includes two functions to prepare texts for analysis. You will choose one or the other for your analysis. The `PrepText` function prepares texts for networks using all types of words, while the `PrepTextNounPhrases` prepares text for networks using only nouns and noun phrases. Users may prefer to create networks based on only nouns or noun phrases because previous studies have shown that such parts of speech are more useful in mapping the topical content of a text than other parts of speech, such as verbs or adjectives (e.g. Rule, Cointet, and Bearman 2015).

PrepText

The `PrepText` function prepares texts for network analysis using part of speech tagging. By default, this function prepares texts for networks using all parts of speech, it can also be limited to nouns and noun compounds. Users may prefer to create networks based on only nouns and noun compounds because previous studies have shown that such parts of speech are more useful in mapping the topical content of a text than other parts of speech, such as verbs or adjectives (e.g. Rule, Cointet, and Bearman 2015).

Let’s begin with the `PrepText` function. This function has three required arguments: (1) `textdata` , a dataframe containing the texts to be analyzed and at least one additional column containing groups; (2) `textvar` , the name of the column containing the texts as a string; (3) `groupvar` , the name of the column containing the the groups through which the words of those texts will be linked as a string. For example, the latter could contain unique document ids, if the user would like to link words co-occurring within documents, or it could be author ids, if the user would like to link words co-occurring by authors, etc. In network analysis terminology, the `textvar` and the `groupvar` are specifying the nodes sets of a two-mode network.

Additionally, the `PrepText` function takes eight optional arguments which control the exact way the text is processed. First, users should specify which type of two-mode network projection they are interested in using the `node_type` argument. `node_type = "words"` prepares texts for a network in which words will be the nodes (with edges to each other based on co-appearance in the same group) and `node_type = "groups"` prepares data for a network with groups as nodes (with edges based on overlap in words between the groups). An example of the former application is Rule, Cointet, and Bearman (2015), and an example of the latter application is Bail (2016).

The other optional arguments are (1) `tokenizer` which controls how words are divided into unit of analysis and allows accurate tokenization of Twitter texts including mentions (@) and hashtags (#) with the specification `tokenizer = "tweets"` ; (2) `pos` which controls whether all parts of speech (`pos = "all"`) or just nouns and noun compouns should be returned (`pos = "nouns"`); (3) `language` which controls the language for part-of-speech tagging and the stop word lexicon; (4) `udmodel_lang` which allows users to pass a previously loaded udpipe model to the function; and (5-7) a set of control arguments specifying whether stop words, i.e. very common nouns such as “and”, “the”, “a”, in English, should be removed (`remove_stop_words`), whether numeric tokens should be deleted (`remove_numbers`); for `tokenizer = "tweets"` only), and whether compound nouns should be identified and returned (`compound_nouns`). The function also allows to pass arguments to the specific tokenizer backend, such as `strip_numeric` for `tokenizer = "words"` or `strip_url` for `tokenizer = "tweets"` .

The output of the `PrepText` function is a dataframe in “tidytext” style, where each row of the dataframe describes a word, the document that it appears in, and its overall frequency within that document.

The following code prepares the first State of the Union address for each president, specifying that nodes will be the group of presidents, with edges drawn according to the overlap of words used in their speeches. In this example we also remove stop words and return noun compounds. Since part-of-speech tagging is a lengthy process, we are only using the first speech for each president to simply our working example:

```
sotu_first_speeches <- sotu %>% group_by(president) %>% slice(1L)
```

On a 2017 MacBook Pro (with a 2.3 GHz i5 & 8 GB ram), the code below a little less than five minutes to run.

```
prepped_sotu <- PrepText(sotu_first_speeches, groupvar = "president", textvar = "sotu_te
xt", node_type = "groups", tokenizer = "words", pos = "nouns", remove_stop_words = TRUE,
 compound_nouns = TRUE)
```

Creating Text Networks

The workhorse function within the textnets package is the CreateTextnet function. This function reads in an object created using the `PrepText` function and outputs an igraph object based on a weighted adjacency matrix, or a square matrix where the rows and columns correspond to either the groups of the group variable (if the user specificed `node_type = "groups"` in the previous stage), or words (if the user specified `node_type = "words"`). The cells of the adjacency matrix are the transposed crossproduce of the term-frequency inverse-document frequency (TFIDF) for overlapping terms between two documents for PrepText and the matrix product of TFIDF crosspropduct (See Bail, 2016).

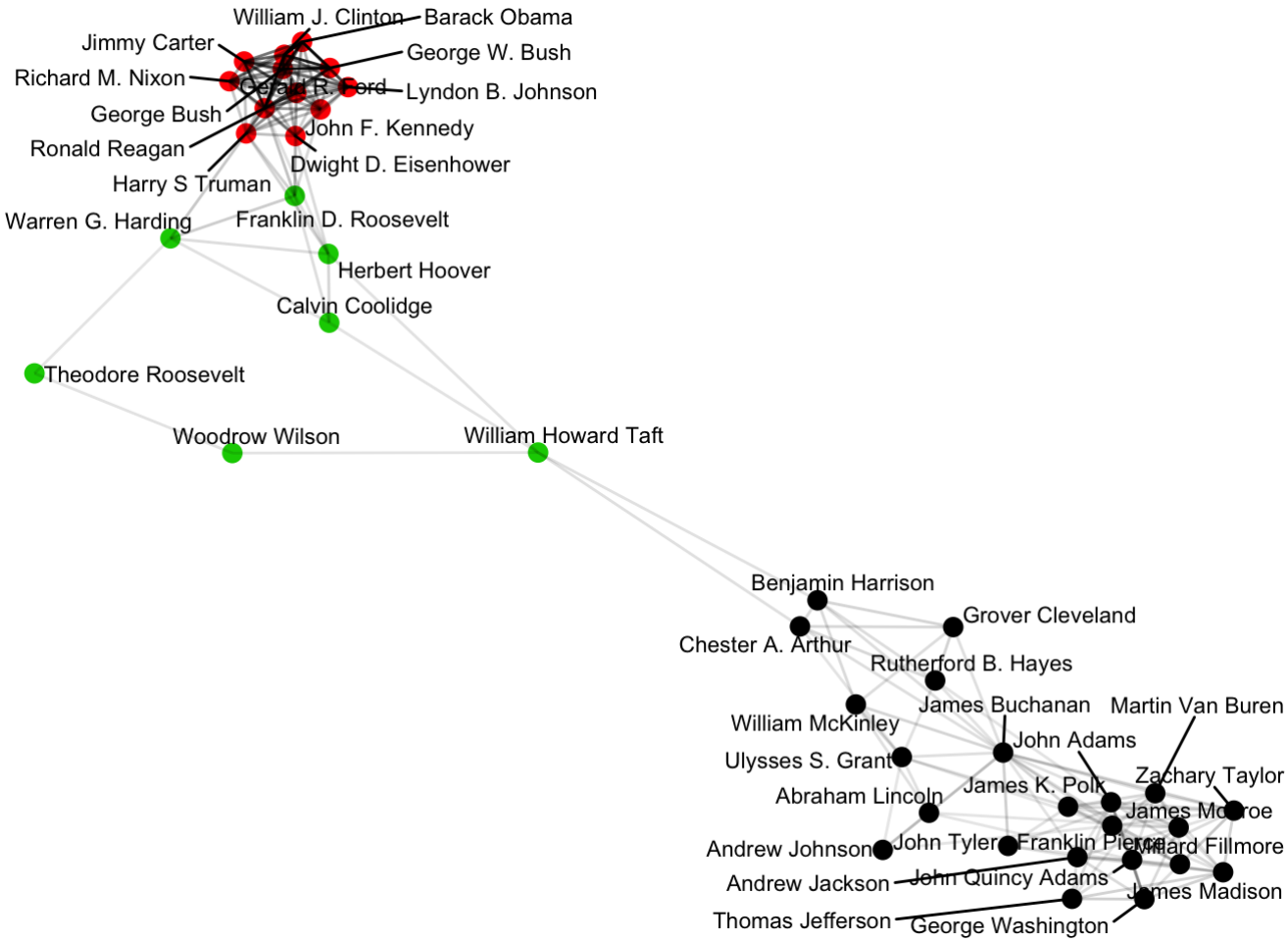
```
sotu_text_network <- CreateTextnet(prepped_sotu)
```

Visualizing Text Networks

The textnets package includes two functions to visualize text networks created in the previous steps. The `VisualizeText` function creates a network diagram where nodes are colored by their cluster or modularity class (see previous section). In many cases, text networks will be very dense (that is, there will be a very large number of edges because most documents share at least one word). Visualizing text networks therefore creates inherent challenges, because such dense networks are very cluttered. To make text networks more readable, the `VisTextNet` function employs a “network backbone” technique which deletes edges using a disparity filter

algorithm to trim edges that are not informative. By default, the function uses a tuning parameter called alpha which is set to .25. The user can specify different levels of alpha to trim more or less of the network. The `VisTextNet` function also includes an argument that determines which nodes will be labeled, since network visualizations with too many node labels can be difficult to interpret. The user specifies an argument called `label_degree_cut` which specifies the degree, or number of connections, that nodes should have to get labeled. For example, if the user only wants nodes that have at least 0 connections to other nodes to be labeled, they would use the following code:

```
VisTextNet(sotu_text_network, label_degree_cut = 0)
```



The user can also specify whether nodes should be sized according to their betweenness centrality using the `betweenness=TRUE` argument. For more details about why a researcher might want to do this see the section entitled “Centrality Measures” below.

The second visualization function in the textnets package is the `VisTextNetD3` function. This function outputs an interactive javascript visualization of the text network, where the user can mouse over each node in order to reveal its node label. Once again, nodes are coloured by their modularity class.

```
VisTextNetD3(sotu_text_network)
```

To save this as an html file for sharing with others or in a presentation, the following can be used. The `height` and `width` parameters are set in pixels, and `bound=TRUE` will prevent the network from dispersing beyond these dimensions. While this may help viewers to see all nodes, it will also cause nodes to cluster at the limits of height

and wigth. This can be prevented by increasing the `charge` parameters, which specifies the strength of node repulsion (negative value) or attraction (positive value). The `zoom` parameter indicates whether to allow users to zoom in and out of the network, which can be especially helpful in large networks for exploring clusters.

In order to save this interative visualization as a .html file, users can use the `htmlwidgets` package as follows:

```
library(htmlwidgets)
vis <- VisTextNetD3(sotu_text_network,
                        prune_cut=.50,
                        height=1000,
                        width=1400,
                        bound=FALSE,
                        zoom=TRUE,
                        charge=-30)
saveWidget(vis, "sotu_textnet.html")
```

Analyzing Text Networks

In order to group documents according to their similarity– or in order to identify latent themes across texts– users may wish to cluster documents or words within text networks. The TextCommunities function applies the Louvain community detection algorithm to do this, which automatically uses the edge weights and determines the number of clusters within a given network. The function outputs a dataframe with the cluster or "modularity" class to which each document or word has been assigned.

```
sotu_communities <- TextCommunities(sotu_text_network)
```

```
##
## Attaching package: 'igraph'
```

```
## The following objects are masked from 'package:dplyr':
##
##     as_data_frame, groups, union
```

```
## The following objects are masked from 'package:stats':
##
##     decompose, spectrum
```

```
## The following object is masked from 'package:base':
##
##     union
```

```
head(sotu_communities)
```

```
##               group modularity_class
## 1   Abraham Lincoln                1
## 2    Andrew Jackson                1
## 3    Andrew Johnson                1
## 4      Barack Obama                2
## 5 Benjamin Harrison                1
## 6   Calvin Coolidge                1
```

In order to further understand which terms are driving the clustering of documents or words, the user can use the InterpretText function, which also reads in an object created by the CreateTextnet function and outputs the words with the 10 highest TFIDF frequencies within each cluster or modularity class. In order to match words, the function requires that the user specify the name of the text data frame object used to create the text network– in this case sotu_text_data (see above).

```
top_words_modularity_classes <- InterpretText(sotu_text_network, prepped_sotu)
head(top_words_modularity_classes)
```

```
## # A tibble: 6 x 2
## # Groups:   modularity_class [2]
##   modularity_class lemma
##   <chr>            <chr>
## 1 2                recovery plan
## 2 1                structure
## 3 2                drug
## 4 2                tonight
## 5 2                budget
## 6 2                medicare
```

Centrality Measures

Often in social networks, researchers wish to calculate measures of influence or centrality in order to predict whether or not occupying brokerage positions can create greater social rewards for individuals. As Bail (2016) shows, the same logic can be applied to text networks to develop a measure of “cultural betweenness” or the extent to which a given document or word is between clusters. To calculate cultural betweennes as well as other centrality measures, textnet users can use the TextCentrality function.

```
text_centrality <- TextCentrality(sotu_text_network)
```

References

Bail, Christopher A. 2016. “Combining Network Analysis and Natural Language Processing to Examine how Advocacy Organizations Stimulate Conversation on Social Media.” *Proceedings of the National Academy of Sciences*, 113:42 11823-11828

Rule, Alix and Jean-Philippe Cointet and Peter Bearman. 2015. “Lexical shifts, substantive changes, and continuity in the State of the Union Discourse, 1790-2014.” *Proceedings of the National Academy of Sciences*