

A Tidy Data Model for Natural Language Processing using cleanNLP

by Taylor Arnold

Abstract The package **cleanNLP** provides a set of fast tools for converting a textual corpus into a set of normalized tables. The underlying natural language processing pipeline utilizes Stanford's CoreNLP library, exposing a number of annotation tasks for text written in English, French, German, and Spanish. Annotators include tokenization, part of speech tagging, named entity recognition, entity linking, sentiment analysis, dependency parsing, coreference resolution, and information extraction.

Introduction

A number of R packages have been developed to construct and manipulate collections of normalized data tables, recently popularized by Hadley Wickham under the term "tidy data" (Wickham et al., 2014), such as **dplyr**, **ggplot2**, **magrittr**, **broom**, **janitor**, and **tidyr**. The functionality provided by the **cleanNLP** package brings this philosophy to the processing of raw textual data by offering three distinct contributions:

- a data schema representing the output of an NLP annotation pipeline as a collection of normalized tables;
- a set of native Java output functions converting a Stanford CoreNLP annotation object directly, without converting into an intermediate XML format, into this collection of normalized tables;
- tools for converting from the tidy model into (sparse) data matrices appropriate for exploratory and predictive modeling.

Together, these contributions simplify the process of doing exploratory data analysis over a corpus of text. The output works seamlessly with both tidy data tools as well as other programming and graphing systems.

The package **cleanNLP** has been designed to integrate into workflows that utilize the many other packages for text processing available in R. Users may use the framework provided by **tm** (Meyer et al., 2008) to manage external corpora or the classes within **NLP** (Hornik, 2016a) to run alternative parsers that can be converted into a tidy framework by way of the `from_ConLL` function. The Apache OpenNLP annotation pipeline, available via **openNLP** (Hornik, 2016b), for instance, provides several languages not yet supported by the Stanford CoreNLP pipeline. Packages that focus on the analysis and modeling of text data can usually be used directly with the output from **cleanNLP**; these include **lda** (Chang, 2015), **lsa** (Wild, 2015), and **topicmodels** (Grün and Hornik, 2011). Similarly, general-purpose database back-ends such as **sqlite** can be used to store the tidy data tables; predictive modeling functions may be used to do predictive analytics over generated term-frequency matrices. In the remainder of this article the basic usage of the **cleanNLP** package is illustrated and the rationale behind the underlying data model is explained. The types of analysis exposed by the model are illustrated by examples from a collection of the text from every State of the Union address made by a United States President.

Java internals and the Stanford CoreNLP pipeline

Wrappers to the Stanford CoreNLP annotation engine are directly provided by **cleanNLP** (Manning et al., 2014). This pipeline is competitively state-of-the-art across all common annotation tasks and supports a relatively large number of natural languages. A key reason for using CoreNLP is that it provides a wider and more complete set of corpus annotations compared to its competitors. Tasks supported by CoreNLP such as tagging OpenIE triples, making speaker annotations, and resolving Wikipedia entity mentions are not included in other commonly used NLP software libraries: SyntaxNet, spaCy, and OpenNLP (Jiang et al., 2016). Given the goal of specifying a tidy data model for representing the output of NLP annotators, it was advantageous to build **cleanNLP** around a pipeline that supports the most complete set of these.

There are two existing R packages that also call functions in the CoreNLP library. The package **StanfordCoreNLP** (Hornik, 2016c), available only through the datacube website at Vienna University, integrates into the **NLP** framework. A similar, standalone approach is offered by **coreNLP** (Arnold and Tilton, 2016). Both of these packages run the annotation pipeline over a corpus of text, call the java class `edu.stanford.nlp.pipeline.XMLOutputter`, and then parse the output using the **XML** package. This approach is not ideal for two reasons. First, parsing the output XML file is computationally

time-consuming. It is also error prone because there is no published format specifying the output of the XML.¹ Secondly, there is a large amount of information lost in the conversion to the XML format. For example, the sentiment parser gives predicted probabilities for each class but the XML output only reports the highest predicted class. Annotators such as the Wikipedia entity resolver and the OpenIE triples parser are not included anywhere in the XML output, making their use impossible through the approach employed in **StanfordCoreNLP** and **coreNLP**. A key intervention of the **cleanNLP** package is the inclusion of custom Java code that saves the output of CoreNLP annotators directly as plain-text, comma separated values. Classes such as `CSVDependencyOutputter` extend the `edu.stanford.nlp.pipeline.AnnotationOutputter` class. The format of these files directly mirrors that of the tidy data model provided by **cleanNLP**. The internal details are unlikely to be of interest to most users of the R package, but the benefits in terms of speed and completeness are critical. As per CRAN policy, the Java source code for these output functions is included in the published **cleanNLP** source tarball.

Package setup

When a corpus of text is annotated by **cleanNLP**, the output can be stored as either a list object containing seven data tables or written to disk as seven plain-text, comma separated value (csv) files. Users who wish to work exclusively with text that has been annotated by others need only to install the package as normal and load it using the `library` command. Many users will eventually wish to annotate their own text; this requires some additional steps to set-up correctly. First of all, two large model .jar files must be downloaded from the Stanford CoreNLP website. This needs to be done only once. There is a convenient wrapper function, `download_clean_nlp`, that downloads the files seamlessly and extracts them (by default) into the local R library directories.

```
> library(cleanNLP)
> download_clean_nlp()
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
   Dload  Upload   Total   Spent    Left     Speed
100  438M  100  438M    0     0  3775k      0  0:01:58  0:01:58 --:--:-- 1797k
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
   Dload  Upload   Total   Spent    Left     Speed
100 1447M  100 1447M    0     0  4458k      0  0:05:32  0:05:32 --:--:-- 5463k
```

The same function has an option to download the specific model files for French, German, and Spanish text. There are also options for downloading from an alternative or local repository in order to save bandwidth given the large file sizes. Next, the language and desired “speed” of the annotation pipeline must be set. This is done with the `set_language` function, with options for English (“en”), Spanish (“es”), French (“fr”), and German (“de”) currently available.

```
> set_language("en", 1)
```

The user’s choice will be saved between sessions, but can be modified at any time. The speed code is a shorthand way of indicating which annotators will be loaded, with higher values of speed indicating more annotators at the expense of a slower run-time. Table 1 details the exact meaning of each speed code. Users familiar with the CoreNLP pipeline may alternatively set properties directly with the `set_property` function. The logic behind these default choices are discussed at length in Section 2.4, in relationship to each data table.

The final step in setting up the **cleanNLP** package involves starting the Java engine and loading the relevant model files. This requires the `init_clean_nlp` function, which will by default print a detailed list of annotators as they are loaded:

```
> init_clean_nlp()
Loading NLP pipeline.
(This may take several minutes. Please be patient.)
[main] INFO edu.stanford.nlp.pipeline.StanfordCoreNLP - Adding annotator tokenize
[main] INFO edu.stanford.nlp.pipeline.StanfordCoreNLP - Adding annotator ssplit
[main] INFO edu.stanford.nlp.pipeline.StanfordCoreNLP - Adding annotator pos
[main] INFO edu.stanford.nlp.pipeline.StanfordCoreNLP - Adding annotator lemma
[main] INFO edu.stanford.nlp.pipeline.StanfordCoreNLP - Adding annotator parse
[main] INFO edu.stanford.nlp.pipeline.StanfordCoreNLP - Adding annotator depparse
```

¹This author, who is also the maintainer of **coreNLP**, has witnessed this first-hand by way of the persistent bug reports centering around the formatting of the XML output in strange edge cases or over new versions of the CoreNLP library.

	runtime	base	parse	sentiment	ner	mention	wiki	triples	coref
Speed 0	1.5 s	X
Speed 1	22 s	X	X	X
Speed 2	35 s	X	X	X	X	X	.	.	.
Speed 3	52 s	X	X	X	X	X	X	.	.
Speed 4	70 s	X	X	X	X	X	.	X	.
Speed 5	91 s	X	X	X	X	X	.	.	X
Speed 6	115 s	X	X	X	X	X	X	.	X
Speed 7	132 s	X	X	X	X	X	.	X	X
Speed 8	156 s	X	X	X	X	X	X	X	X

Table 1: Available speed codes and their associated annotators. The base level includes sentence splitting, tokenization, and part of speech tagging. Setting above speed 1 has no additional effect for the French models; setting above speed 2 has no additional effect for German and Spanish models. The times are the median run time to parse a 4000-word English document across 5 replications.

```
[main] INFO edu.stanford.nlp.pipeline.StanfordCoreNLP - Adding annotator sentiment
NLP pipeline finished loading.
```

As speed code 1 has been selected, loading the annotators takes only a few seconds. Higher speed codes may take upwards of several minutes to load all of the models into memory. Importantly, this step will fail if R is not able to find Java version 1.8 on the machine. The documentation for [rjava](#) should be consulted for help with any difficulties. During an active R session, it is possible to modify the language or speed codes. The pipeline must be reinitialized, however, in order for changes to take effect.

Once the pipeline is initialized, textual data can be processed using the function `annotate`. It takes either a set of file paths for the location of the input text files (the default), or character vectors to be parsed directly in R. The output consists of either plain-text files written to the disk, a returned R list object of class `annotation`, or an R list object of class `annotation` written in binary format directly to the file system. For example, data used in this article came from the following:

```
> annotate("/path/to/corpus/*.txt", file = "sotu.Rds", load = FALSE)
```

By default each individual file, or element in a character vector if passing directly as strings, is treated as its own document. As further explained in Section 2.4.1, helper functions exist to combine annotation objects (`combine_documents`) and to extract specific documents from an annotation object (`extract_documents`).

A data model for the NLP pipeline

An annotation object is simply a named list with each item containing a data frame. These frames should be thought of as tables living inside of a single database, with keys linking each table to one another. All tables are in the second normal form of [Codd \(1990\)](#). For the most part they also satisfy the third normal form, or, equivalently, the formal tidy data model of [Wickham et al. \(2014\)](#). The limited departures from this more stringent requirement are justified below wherever they exist. In every case the cause is a transitive dependency that would require a complex range join to reconstruct.

Several standards have previously been proposed for representing textual annotations. These include the linguistic Annotation Framework ([Ide and Romary, 2001](#)), NLP Interchange Format ([Hellmann et al., 2012](#)), and CoNLL-X ([Buchholz and Marsi, 2006](#)).² The function `from_ConLL` is included as a helper function in `cleanNLP` to convert from CoNLL formats into the `cleanNLP` data model. All of these, however, are concerned with representing annotations for interoperability between systems. Our goal is instead to create a data model well-suited to direct analysis, and therefore requires a new approach.

In this section each table is presented and justifications for its existence and form are given. Individual tables may be pulled out with access functions of the form `get_*`. Example tables are pulled from the State of the Union corpus, which will be discussed at length in the next section.

²The R package [tidytext](#) also claims to have defined a tidy data format for text, but differs from the goals here as it focuses only on the task of tokenization and fails to satisfy the second normal form assumption.

get_document()	
id	integer. Id of the source document.
time	date time. The time at which the parser was run on the text.
version	character. Version of the CoreNLP library used to parse the text.
language	character. Language of the text, in ISO 639-1 format.
uri	character. Description of the raw text location.

Table 2: Schema for the document table. The id field serves as a primary key, and other meta data fields may be appended that give domain-specific information about each document.

Documents

The documents table contains one row per document in the annotation object. What exactly constitutes a document up to the user. It might include something as granular as a paragraph or as coarse as an entire novel. For many applications, particularly stylometry, it may be useful to simultaneously work with several hierarchical levels: sections, chapters, and an entire body of work. The solution in these cases is to define a document as the smallest unit of measurement, denoting the higher-level structures as metadata. The primary key for this table is a document id, stored as an integer index. By design, there should be no extrinsic meaning placed on this key. Other tables use it to map to one another and to the document table, but any metadata *about* the document is contained only in the document table rather than being forced into the document key. In other words, the temptation to use keys such as “Obama2016” is avoided because, while these look nice, they ultimately make programming hard.

The minimal fields required by the document table are given in Table 2. These are all filled in automatically by the annotation function. Any number of additional corpora-specific metadata, such as the aforementioned section and chapter designations, may be attached as well. The document table for the example corpus is:

```
> get_document(sotu)
# A tibble: 236 x 11
   id      time version language  file      president  year
<int>   <dtm>   <chr>   <chr>   <chr>      <chr>   <int>
1     0 2016-11-08 20:02:00  3.7.0    en 239.txt George Washington 1790
2     1 2016-11-08 20:02:00  3.7.0    en 240.txt George Washington 1790
3     2 2016-11-08 20:02:00  3.7.0    en 241.txt George Washington 1791
4     3 2016-11-08 20:03:00  3.7.0    en 242.txt George Washington 1792
5     4 2016-11-08 20:01:00  3.7.0    en 235.txt George Washington 1793
6     5 2016-11-08 20:01:00  3.7.0    en 236.txt George Washington 1794
7     6 2016-11-08 20:02:00  3.7.0    en 237.txt George Washington 1795
8     7 2016-11-08 20:02:00  3.7.0    en 238.txt George Washington 1796
9     8 2016-11-08 20:01:00  3.7.0    en 231.txt      John Adams 1797
10    9 2016-11-08 20:01:00  3.7.0    en 232.txt      John Adams 1798
# ... with 226 more rows, and 4 more variables: years_active <chr>, url <chr>,
# party <chr>, sotu_type <chr>
```

Notice that metadata such as the president, year, and party into the document table has been included. It may seem that common fields such as year and author should be added to the formal specification but the perceived advantage is minimal. It would still be necessary for users to manually add the content of these fields at some point as any other metadata is not unambiguously extractable from the raw text.

Tokens

The token table contains one row for each unique *token*, usually a word or punctuation mark, in any document in the corpus. Any annotator that produces an output for each token has its results displayed here. These include the lemmatizer, the part of the speech tagger (Toutanova and Manning, 2000) and speaker indicators (Toutanova et al., 2003). Table 3 shows the required columns contained in the token table. Given the annotators selected during the pipeline initialization, some of these columns may contain only missing data. A composite key exists by taking together the document id, sentence id, and token id. There is also a set of foreign keys cid and cid_end giving character offsets back into the original source document. An example of the table looks like this:

```
> get_token(sotu)
# A tibble: 2,212,766 x 11
```

get_token()	
id	integer. Id of the source document.
sid	integer. Sentence id, starting from 0.
tid	integer. Token id, with the root of the sentence starting at 0.
word	character. Raw word in the input text.
lemma	character. Lemmatized form the token.
upos	character. Universal part of speech code.
pos	character. Language-specific part of speech code; uses the Penn Treebank codes.
speaker	character. Identity of the speaker, with PER0 denoting the narratorial voice.
wiki	character. Link to Wikipedia entity.
cid	integer. Character offset at the start of the word in the original document.
cid_end	integer. Character offset pointing one past the character at the end of the word.

Table 3: Schema for the token table. The fields id, sid, and tid serve as a composite key for each token. A row also exist for the root of each sentence.

	id	sid	tid	word	lemma	upos	pos	speaker
	<int>	<int>	<int>	<chr>	<chr>	<chr>	<chr>	<chr>
1	0	0	0	ROOT	ROOT	<NA>	<NA>	<NA>
2	0	0	1	Fellow-Citizens	fellow-citizen	NOUN	NNS	PER0
3	0	0	2	of	of	ADP	IN	PER0
4	0	0	3	the	the	DET	DT	PER0
5	0	0	4	Senate	Senate	NOUN	NNP	PER0
6	0	0	5	and	and	CONJ	CC	PER0
7	0	0	6	House	House	NOUN	NNP	PER0
8	0	0	7	of	of	ADP	IN	PER0
9	0	0	8	Representatives	Representatives	NOUN	NNPS	PER0
10	0	0	9	:	:	.	:	PER0
# ... with 2,212,756 more rows, and 3 more variables: wiki <chr>, cid <int>, # cid_end <int>								

A phantom token “ROOT” is included at the start of each sentence (it always has tid equal to 0). This was added so that joins from the dependency table, which contains references to the sentence root, into the token table have no missing values.

The field upos contains the universal part of speech code, a language-agnostic classification, for the token. It could be argued that in order to maintain database normalization one should simply look up the universal part of speech code by finding the language code in the document table and joining a table mapping the Penn Treebank codes to the universal codes. This has not been done for several reasons. First, universal parts of speech are very useful for exploratory data analysis as they contain tags much more familiar to non-specialists such as “NOUN” (noun) and “CONJ” (conjunction). Asking users to apply a three table join just to access them seems overly cumbersome. Secondly, it is possible for users to use other parsers or annotation engines. These may not include granular part of speech codes and it would be difficult to figure out how to represent these if there were not a dedicated universal part of speech field.

Dependencies

Dependencies give the grammatical relationship between pairs of tokens within a sentence (Green et al., 2011; Rafferty and Manning, 2008). As they are at the level of token pairs, they must be represented as a new table. All included fields are described in Table 4. Only one dependency should exist for any pair of tokens; the document id, sentence id, and source and target token ids together serve as a composite key. As dependencies exist only within a sentence, the sentence id does not need to be defined separately for the source and target. Dependencies take significantly longer to calculate than the lemmatization and part of speech tagging tasks. By default, the set_language function selects a fast neural network parser that requires more memory but runs nearly twice as fast as other default parsers in the CoreNLP pipeline (Chen and Manning, 2014).

The get_dependency function has an option (set to FALSE by default) to auto join the dependency to the target and source tokens and words from the token table. This is a common task and involves non-trivial calls to the left_join function making it worthwhile to include as an option. The output, with the option turned on, is given by:

get_dependency()	
id	integer. Id of the source document.
sid	integer. Sentence id of the source token.
tid	integer. Id of the source token.
sid_target	integer. Sentence id of the target token.
tid_target	integer. Id of the target token.
relation	character. Language-agnostic universal dependency type.
relation_full	character. Language specific universal dependency type.
word	character. The source word in the raw text.
lemma	character. Lemmatized form of the source word.
word_target	character. The target word in the raw text.
lemma_target	character. Lemmatized form of the target word.

Table 4: Schema for the dependency table. The final four variables are only provided when the option `get_token` is set to `TRUE`. The first five fields together create a composite key for the table.

```
> get_dependency(sotu, get_token = TRUE)
Joining, by = c("id", "sid", "tid")
Joining, by = c("id", "sid_target", "tid_target")
# A tibble: 2,266,444 x 11
  id sid tid sid_target tid_target relation relation_full word
  <int> <int> <int> <int> <int> <chr> <chr> <chr>
1 0 0 0 0 1 root root ROOT
2 0 0 4 0 2 case case Senate
3 0 0 4 0 3 det det Senate
4 0 0 1 0 4 nmod nmod:of Fellow-Citizens
5 0 0 4 0 5 cc cc Senate
6 0 0 1 0 6 nmod nmod:of Fellow-Citizens
7 0 0 4 0 6 conj conj:and Senate
8 0 0 8 0 7 case case Representatives
9 0 0 6 0 8 nmod nmod:of House
10 0 0 1 0 9 punct punct Fellow-Citizens
# ... with 2,266,434 more rows, and 3 more variables: lemma <chr>, word_target <chr>,
# lemma_target <chr>
```

The word “ROOT” shows up in the first row, which would have been NA had sentence roots not been explicitly included in the token table.

Our parser produces universal dependencies (De Marneffe et al., 2014), which have a language-agnostic set of relationship types with language-specific subsets pertaining to specific grammatical relationships with a particular language. For the same reasons that both the part of speech codes and universal part of speech codes are included, each of these relationship types have been added to the dependency table.

Coreference

Coreferences link sets of tokens that refer to the same underlying person, object, or idea (Recasens et al., 2013; Lee et al., 2013, 2011; Raghunathan et al., 2010). One common example is the linking of a noun in one sentence to a pronoun in the next sentence. The coreference table describes these relationships but is not strictly a table of coreferences. Instead, each row represents a single mention of an expression and gives a reference id indicating all of the other mentions that it also coreferences. In theory, a given set of tokens might have two separate mentions if it refers to two different classes of references (though this is quite rare). Table 5 gives the entire schema of the coreference table. The document, reference, and mention ids serve as a composite key for the table. Links back into the token table for the start, end and head of the mention are given as well; these are pushed to the right of the table as they should be considered foreign keys within this table.

An example helps to explain exactly what the coreference table represents (the first row is removed as its mention is quite long and makes the table hard to read):

```
> get_coreference(sotu)[-1,]
# A tibble: 179,167 x 12
  id rid mid mention mention_type number gender animacy
```


get_coreference()	
id	integer. Id of the source document.
rid	integer. Relation ID.
mid	integer. Mention ID; unique to each coreference within a document.
mention	character. The mention as raw words from the text.
mention_type	character. One of "LIST", "NOMINAL", "PRONOMINAL", or "PROPER".
number	character. One of "PLURAL", "SINGULAR", or "UNKNOWN".
gender	character. One of "FEMALE", "MALE", "NEUTRAL", or "UNKNOWN".
animacy	character. One of "ANIMATE", "INANIMATE", or "UNKNOWN".
sid	integer. Sentence id of the coreference.
tid	integer. Token id at the start of the coreference.
tid_end	integer. Token id at the start of the coreference.
tid_head	integer. Token id of the head of the coreference.

Table 5: Schema for the coreference table. Each row is best thought of as a coreference mention, rather than the coreference itself.

get_entity()	
id	integer. Id of the source document.
sid	integer. Sentence id of the entity mention.
tid	integer. Token id at the start of the entity mention.
tid_end	integer. Token id at the end of the entity mention.
entity_type	character. Type of entity.
entity	character. Raw words of the named entity in the text.
entity_normalized	character. Normalized version of the entity.

Table 6: Schema for the entity table. The first three fields serve as a composite key.

	<int>	<int>	<int>		<chr>	<chr>	<chr>	<chr>	<chr>
1	0	257	257	the public credit	NOMINAL	SINGULAR	NEUTRAL	INANIMATE	
2	0	195	193	the measures of government	NOMINAL	PLURAL	UNKNOWN	INANIMATE	
3	0	195	195	their	PRONOMINAL	PLURAL	UNKNOWN	ANIMATE	
4	0	291	13	our public affairs	NOMINAL	PLURAL	UNKNOWN	INANIMATE	
5	0	291	291	the affairs	NOMINAL	PLURAL	UNKNOWN	INANIMATE	
6	0	39	10	you	PRONOMINAL	UNKNOWN	UNKNOWN	ANIMATE	
7	0	39	45	your	PRONOMINAL	UNKNOWN	UNKNOWN	ANIMATE	
8	0	39	34	you	PRONOMINAL	UNKNOWN	UNKNOWN	ANIMATE	
9	0	39	39	your	PRONOMINAL	UNKNOWN	UNKNOWN	ANIMATE	
10	0	39	44	you	PRONOMINAL	UNKNOWN	UNKNOWN	ANIMATE	
# ... with 179,157 more rows, and 4 more variables: sid <int>, tid <int>, tid_end <int>, # tid_head <int>									

There is a special relationship between the reference id *rid* and the mention id *mid*. The coreference annotator selects a specific mention for each reference that gets treated as the canonical mention for the entire class. The mention id for this mention becomes the reference id for the class, as can be in the above table with rows 1, 3, 5, and 9 all corresponding to the canonical mention of their respective classes. This relationship provides a way of identifying the canonical mention within a reference class and a way of treating the coreference table as pairs of mentions rather than individual mentions joined by a given key.

The text of the mention itself is included within the table. This was done because as the mention may span several tokens it would otherwise be very difficult to extract this information from the token table. It is also possible, though not supported in the current CoreNLP pipeline, that a mention could consist of a set of non-contiguous tokens, making this field impossible to otherwise reconstruct.

Named entities

Named entity recognition is the task of finding entities that can be defined by proper names, categorizing them, and standardizing their formats (Finkel et al., 2005). The XML output of the Stanford CoreNLP pipeline places named entity information directly into their version of the token table. Doing

get_sentiment()	
id	integer. Id of the source document.
sid	integer. Sentence id.
pred_class	integer. Predicted sentiment class of the sentence, from 0 (worst) to 4 (best).
p0	double. Predicted probability of the sentence being class 0.
p1	double. Predicted probability of the sentence being class 1.
p2	double. Predicted probability of the sentence being class 2.
p3	double. Predicted probability of the sentence being class 3.
p4	double. Predicted probability of the sentence being class 4.

Table 7: Schema for the sentiment table. The document and sentence ids serve as a composite key.

this repeats information over every token in an entity and gives no canonical way of extracting the entirety of a single entity mention. We instead have a separate entity table, as is demanded by the normalized database structure, and record each entity mention in its own row. The full set of fields are given in Table 6, with the combination of document id, sentence id, and token id serving as a composite key.

An example of the named entity table is given by:

```
> get_entity(sotu)
# A tibble: 94,564 x 7
      id  sid  tid tid_end entity_type entity entity_normalized
  <int> <int> <int>   <int>   <chr>   <chr>         <chr>
1     0    0    4     4 ORGANIZATION Senate          <NA>
2     0    0    6     8 ORGANIZATION House of Representatives <NA>
3     0    0   18    18      DATE now PRESENT_REF
4     0    0   26    26      DATE present PRESENT_REF
5     0    1    9    10 LOCATION North Carolina <NA>
6     0    1   16    17 LOCATION United States <NA>
7     0    3   27    27      DATE present PRESENT_REF
8     0    5    7     7      NUMBER one 1.0
9     0    9   18    18      MISC Indians <NA>
10    0    9   62    62 LOCATION Virginia <NA>
# ... with 94,554 more rows
```

The categories available in the field `entity_type` are dependent on the models used in the annotation pipeline. The default English model selected for speed codes 2 and above include the categories: “LOCATION”, “PERSON”, “ORGANIZATION”, “MISC”, “MONEY”, “PERCENT”, “DATE” and “TIME”. The last four of these also have a normalized form, given in the final field of the table. As with the coreference table, a complete representation of the entity is given as a character string due to the difficulty in reconstructing this after the fact from the token table.

Sentiment analysis

The *sentiment* tagger provided by the CoreNLP pipeline predicts whether a sentence is very negative (0), negative (1), neutral (2), positive (3), or very positive (4) (Socher et al., 2013). The sentiment output is placed in a separate table because it returns information exclusively at the sentence level, unlike any of the other parsers. The schema is given in Table 7, with the document and sentence ids serving as composite keys. An example of the output can be seen in:

```
> get_sentiment(sotu)
# A tibble: 68,756 x 8
      id  sid pred_class    p0    p1    p2    p3    p4
  <int> <int>   <int>   <dbl> <dbl> <dbl> <dbl> <dbl>
1     0    0         3 0.059697 0.164863 0.105306 0.478228 0.191905
2     0    1         4 0.003425 0.004395 0.037452 0.440832 0.513895
3     0    2         1 0.226266 0.575373 0.164053 0.022456 0.011853
4     0    3         3 0.024617 0.087265 0.163085 0.607931 0.117102
5     0    4         3 0.014715 0.040212 0.101467 0.682611 0.160995
6     0    5         3 0.021507 0.065547 0.200448 0.541503 0.170996
7     0    6         1 0.236505 0.509930 0.238690 0.009211 0.005664
8     0    7         3 0.032885 0.158260 0.278656 0.455413 0.074786
```


	get_triple()
id	integer. Id of the source document.
subject	character. Raw text of the triple's subject.
object	character. Raw text of the triple's object.
relation	character. Raw text of the triple's relation.
confidence	double. Confidence score from 0 (least confident) to 1 (most confident).
be_prefix	integer. Equals 1 if the triple's relationship has a form of "to be" as a prefix.
be_suffix	integer. Equals 1 if the triple's relationship has a form of "to be" as a suffix.
of_suffix	integer. Equals 1 if the triple's relationship has a form of "of" as a suffix.
tmod	integer. Equals 1 if the triple is a temporal modifier.
sid	integer. Sentence id of the triple.
tid_subject	integer. Token id at the start of the triple's subject.
tid_subject_end	integer. Token id at the end of the triple's subject.
tid_object	integer. Token id at the start of the triple's object.
tid_object_end	integer. Token id at the end of the triple's object.
tid_relation	integer. Token id at the start of the triple's relation.
tid_relation_end	integer. Token id at the end of the triple's relation.

Table 8: Schema for the triple table. The default uses the OpenIE triples model, but other such as the KBT are also available and representable in the same form.

```

9      0      8      3 0.010174 0.036523 0.115024 0.734460 0.103818
10     0      9      1 0.194770 0.530428 0.229339 0.027968 0.017495
# ... with 68,746 more rows

```

The underlying model is a neural network. The most likely categorization is included along with the predicted probabilities for all 5 classes.

Information extraction

The final table in the data model contains the information of the *relationship triples* generated (by default) by the OpenIE parser (Angeli et al., 2015). These triples contain a subject, object, and relation, all of which consist of a set of tokens in the raw text. A full description of the schema can be found in Table 8. There are a relatively large number of fields given the need to index three separate phrases that may be scattered across multiple sentences. This is also the only table for which there is no natural composite key (though all of the ids together are guaranteed to be unique).

The following serves as a typical example of the output:

```

> get_triple(anno)
# A tibble: 507,991 x 16
   id      subject      object      relation confidence
  <int>      <chr>      <chr>      <chr>      <dbl>
1     0 rising credit    auspicious    are          1
2     0 accession      auspicious    are          1
3     0 credit          auspicious    are          1
4     0 recent accession auspicious    are          1
5     0 particular regard common defense providing for 1
6     0 they          such manufactories promote      1
7     0 they          render        tend        1
8     0 plan          requisite      is          1
9     0 they render for essential supplies    tend        1
10    0 uniform plan    requisite      is          1
# ... with 507,981 more rows, and 11 more variables: be_prefix <int>,
# be_suffix <int>, of_suffix <int>, tmod <int>, sid <int>, tid_subject <int>,
# tid_subject_end <int>, tid_object <int>, tid_object_end <int>,
# tid_relation <int>, tid_relation_end <int>

```

As with the coreference and entity tables, the complete subject, object, and relation phrases are included given the difficulty of extracting them from the data.

Using cleanNLP to study State of the Union addresses

In this final section the utility of the package is illustrated by showing how it can be used to study a corpus consisting of every State of the Union Address made by a United States president through 2016 (Peters). It highlights some of the major benefits of the tidy datamodel as it applies to the study of textual data, though by no means attempts to give an exhaustive coverage of all the available tables and approaches. The examples make heavy use of the table verbs provided by **dplyr**, the piping notation of **magrittr** and **ggplot2** graphics. These are used because they best illustrate the advantages of the tidy data model that has been built in **cleanNLP** for representing corpus annotations.

Exploratory analysis

Simple summary statistics are easily computed off of the token table. To see the distribution of sentence length, the token table is grouped by the document and sentence id and the number of rows within each group are computed. The percentiles of these counts give a quick summary of the distribution.

```
> get_token(sotu) %>%
+   group_by(id, sid) %>%
+   summarize(sent_len = n()) %$%
+   quantile(sent_len, seq(0,1,0.1))
  0%  10%  20%  30%  40%  50%  60%  70%  80%  90% 100%
  2   12   16   20   24   28   32   37   45   57  679
```

The median sentence has 28 tokens, whereas at least one has over 600 (this is due to a bulleted list in one of the written addresses being treated as a single sentence) To see the most frequently used nouns in the dataset, the token table is filtered on the universal part of speech field, grouped by lemma, and the number of rows in each group are once again calculated. Sorting the output and selecting the top 42 nouns, yields a high level summary of the topics of interest within this corpus.

```
> get_token(sotu) %>%
+   filter(upos == "NOUN") %>%
+   group_by(lemma) %>%
+   summarize(count = n()) %>%
+   top_n(n = 42, count) %>%
+   arrange(desc(count)) %>%
+   use_series(lemma)
[1] "year"      "government" "States"      "Congress"    "United"      "country"
[7] "people"    "law"         "nation"      "time"        "power"       "interest"
[13] "war"       "world"       "citizen"     "service"     "duty"        "system"
[19] "part"      "state"       "peace"       "program"     "man"         "America"
[25] "policy"    "condition"   "act"         "work"        "legislation" "force"
[31] "effort"    "treaty"     "purpose"     "Federal"     "business"    "land"
[37] "subject"   "action"     "Department"  "measure"     "way"         "tax"
```

The result is generally as would be expected from a corpus of government speeches, with references to proper nouns representing various organizations within the government and non-proper nouns indicating general topics of interest such as "tax", "law", and "peace".

The length in tokens of each address is calculated similarly by grouping and summarizing at the document id level. The results can be joined with the document table to get the year of the speech and then piped in a **ggplot2** command to illustrate how the length of the State of the Union has changed over time.

```
> get_token(sotu) %>%
+   group_by(id) %>%
+   summarize(n = n()) %>%
+   left_join(get_document(sotu)) %>%
+   ggplot(aes(year, n)) +
+     geom_line(color = grey(0.8)) +
+     geom_point(aes(color = sotu_type)) +
+     geom_smooth()
Joining, by = "id"
```

Here, color is used to represent whether the address was given as an oral address or a written document. The output in Figure 1 shows that there are certainly time trends to the address length, with the form of the address (written versus spoken) also having a large effect on document length.

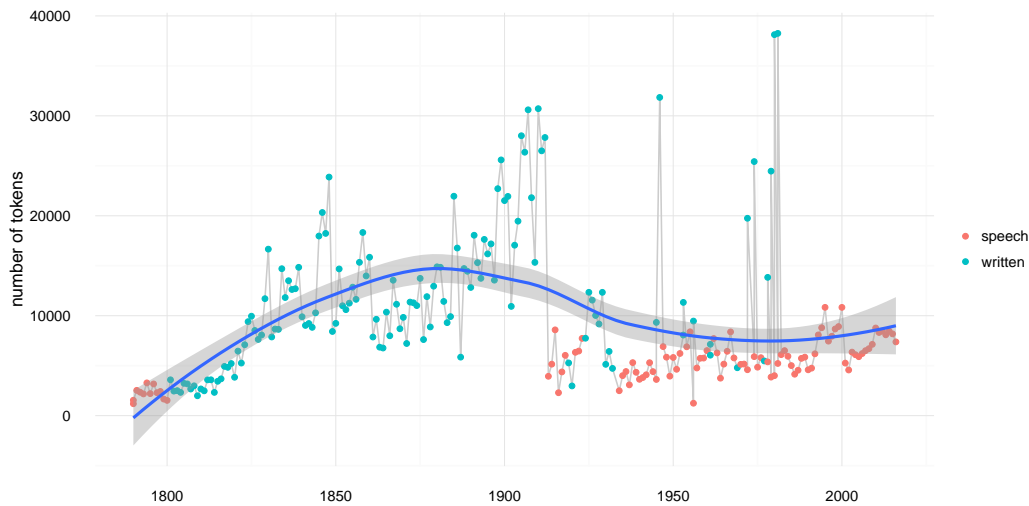


Figure 1: Length of each State of the Union address, in total number of tokens. Color shows whether the address was given as a speech or delivered as a written document.

Finding the most used entities from the entity table over the time period of the corpus yields an alternative way to see the underlying topics. A slightly modified version of the code snippet used to find the top nouns in the dataset can be used to find the top entities. The `get_token` function is replaced by `get_entity` and the table is filtered on `entity_type` rather than the universal part of speech code.

```
> get_entity(sotu) %>%
+   filter(entity_type == "LOCATION") %>%
+   group_by(entity) %>%
+   summarize(count = n()) %>%
+   top_n(n = 44, n) %>%
+   arrange(desc(n)) %>%
+   use_series(entity)
```

[1] "United States"	"America"	"States"	"Mexico"
[5] "Great Britain"	"Spain"	"Europe"	"China"
[9] "Washington"	"France"	"Cuba"	"Texas"
[13] "Japan"	"Pacific"	"Russia"	"Republic"
[17] "Soviet Union"	"Germany"	"Nicaragua"	"California"
[21] "District of Columbia"	"Mississippi"	"Alaska"	"Asia"
[25] "Africa"	"Iraq"	"Atlantic"	"New York"
[29] "U.S."	"Panama"	"Philippines"	"Middle East"
[33] "Canada"	"Afghanistan"	"Paris"	"Brazil"
[37] "Central America"	"Kansas"	"Oregon"	"Iran"
[41] "Peru"	"Italy"	"London"	"Colombia"
[45] "Korea"	"Venezuela"		

The ability to redo analyses from a slightly different perspective is a direct consequence of the tidy data model supplied by **cleanNLP**. The top locations include some obvious and some less obvious instances. Those sovereign nations included such as Great Britain, Mexico, Germany, and Japan seem as expected given either the United State's close ties or periods of war with them. The top states include the most populous regions (New York, California, and Texas) but also smaller states (Kansas, Oregon, Mississippi), the latter being more surprising.

One of the most straightforward way of extracting a high-level summary of the content of a speech is to extract all direct object object dependencies where the target noun is not a very common word. In order to do this for a particular speech, the dependency table is joined to the document table, a particular document is selected, and relationships of type "dobj" (direct object) are filtered out. The result is then joined to the data set `word_frequency`, which is included with **cleanNLP**, and pairs with a target occurring less than 0.5% of the time are selected to give the final result. Here is an example of this using the first address made by George W. Bush in 2001:

```
> get_dependency(sotu, get_token = TRUE) %>%
+   left_join(get_document(sotu)) %>%
```

```

+ filter(year == 2001) %>%
+ filter(relation == "dobj") %>%
+ select(id = id, start = word, word = lemma_target) %>%
+ left_join(word_frequency) %>%
+ filter(frequency < 0.0005) %>%
+ select(id, start, word) %$$
+ sprintf("%s => %s", start, word)
Joining, by = c("id", "sid", "tid")
Joining, by = c("id", "sid_target", "tid_target")
Joining, by = "id"
Joining, by = "word"
[1] "take => oath" "help => disadvantaged"
[3] "fight => homelessness" "fight => illiteracy"
[5] "end => profiling" "throw => darts"
[7] "promoting => internationalism" "makes => downpayment"
[9] "discard => relic" "sound => footing"
[11] "minding => manners"

```

Most of these phrases correspond with the “compassionate conservatism” that George W. Bush ran under in the preceding 2000 election. Applying the same analysis to the 2002 State of the Union, which came under the shadow of the September 11th terrorist attacks, shows a drastic shift in focus.

```

> get_dependency(sotu, get_token = TRUE) %>%
+ left_join(get_document(sotu)) %>%
+ filter(year == 2002) %>%
+ filter(relation == "dobj") %>%
+ select(id = id, start = word, word = lemma_target) %>%
+ left_join(word_frequency) %>%
+ filter(frequency < 0.0005) %>%
+ select(id, start, word) %$$
+ sprintf("%s => %s", start, word)
Joining, by = c("id", "sid", "tid")
Joining, by = c("id", "sid_target", "tid_target")
Joining, by = "id"
Joining, by = "word"
[1] "urged => follower" "brought => sorrow" "hold => hostage"
[4] "eliminate => parasite" "flaunt => hostility" "make => agile"
[7] "fight => anthrax" "equip => firefighter" "defeat => recession"
[10] "want => paycheck" "enact => safeguard" "embracing => ethic"
[13] "owns => aspiration" "containing => resentment" "erasing => rivalry"
[16] "embrace => tyranny"

```

Here the topics have almost entirely shifted to counter-terrorism and national security efforts.

Models

The `get_tfidf` function provided by **cleanNLP** converts a token table into a sparse matrix representing the term-frequency inverse document frequency matrix (or any intermediate part of that calculation). This is particularly useful when building models from a textual corpus. The `tidy_pca`, also included with the package, takes a matrix and returns a data frame containing the desired number of principal components. Dimension reduction involves piping the token table for a corpus into the `get_tfidf` function and passing the results to `tidy_pca`.

```

> pca <- get_token(sotu) %>%
+ filter(pos %in% c("NN", "NNS")) %>%
+ get_tfidf(min_df = 0.05, max_df = 0.95, type = "tfidf", tf_weight = "dnorm") %$$
+ tidy_pca(tfidf, get_document(sotu))

```

In this example only non-proper nouns have been included in order to minimize the stylistic attributes of the speeches in order to focus more on their content. A scatter plot of the speeches using these components is shown in Figure 2. There is a definitive temporal pattern to the documents, with the 20th century addresses forming a distinct cluster on the right side of the plot.

The output of the `get_tfidf` function may be given directly to the LDA function in the package **topicmodels**. The topic model function requires raw counts, so the `type` variable in `get_tfidf` is set to “tf”; the results may then be directly piped to LDA.

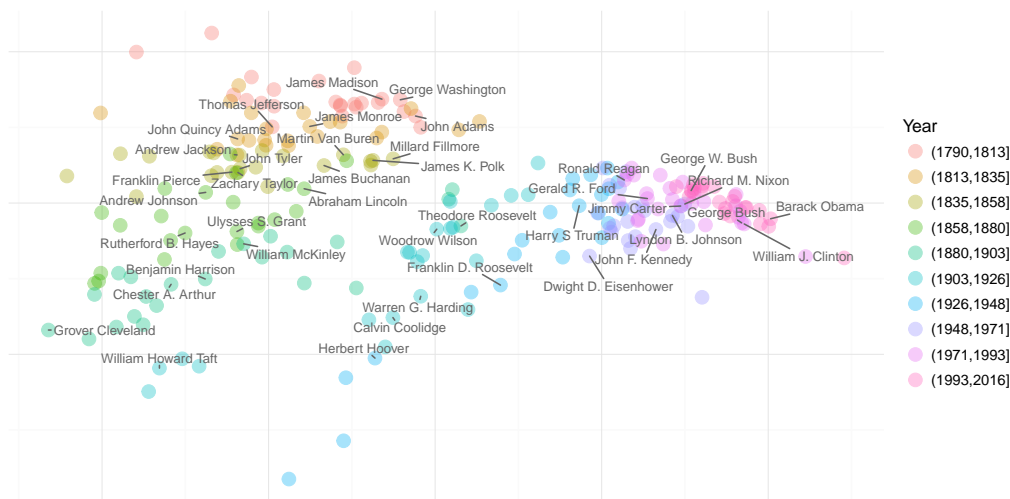


Figure 2: State of the Union Speeches, highlighting each President's first address, plotted using the first two principal components of the term frequency matrix of non-proper nouns.

```
> library(topicmodels)
> tm <- get_token(sotu) %>%
+   filter(pos %in% c("NN", "NNS")) %>%
+   get_tfidf(min_df = 0.05, max_df = 0.95, type = "tf", tf_weight = "raw") %$%
+   LDA(tf, k = 16, control = list(verbose = 1))
```

The topics, ordered by approximate time period, are visualized in Figure 3. Most topics persist for a few decades and then largely disappear, though some persist over non-contiguous periods of the presidency. The Energy topic, for example, appears during the 1950s and crops up again during the energy crisis of the 1970s. The “world, man, freedom, force, defense” topic peaks during both World Wars, but is absent during the 1920s and early 1930s.

Finally, the **cleanNLP** data model is also convenient for building predictive models. The State of the Union corpus does not lend itself to an obviously applicable prediction problem. A classifier that distinguishes speeches made by George W. Bush and Barrack Obama will be constructed here for the purpose of illustration. As a first step, a term-frequency matrix is extracted using the same technique as was used with the topic modeling function. However, here the frequency is computed for each sentence in the corpus rather than the document as a whole. The ability to do this seamlessly with a single additional mutate function defining a new id illustrates the flexibility of the `get_tfidf` function.

```
> df <- get_token(sotu) %>%
+   left_join(get_document(sotu)) %>%
+   filter(year > 2000) %>%
+   mutate(new_id = paste(id, sid, sep = "-")) %>%
+   filter(pos %in% c("NN", "NNS"))
Joining, by = "id"
> mat <- get_tfidf(df, min_df = 0, max_df = 1, type = "tf",
+                 tf_weight = "raw", doc_var = "new_id")
```

It will be necessary to define a response variable y indicating whether this is a speech made by President Obama as well as a training flag indicating which speeches were made in odd numbered years. This is done via a separate table join and a pair of mutations.

```
> meta <- data_frame(new_id = mat$id) %>%
+   left_join(df[!duplicated(df$new_id),]) %>%
+   mutate(y = as.numeric(president == "Barack Obama")) %>%
+   mutate(train = year %in% seq(2001,2016, by = 2))
Joining, by = "new_id"
```

The output may now be used as input to the elastic net function provided by the **glmnet** package. The response is set to the binomial family given the binary nature of the response and training is done on only those speeches occurring in odd-numbered years. Cross-validation is used in order to select the best value of the model's tuning parameter.

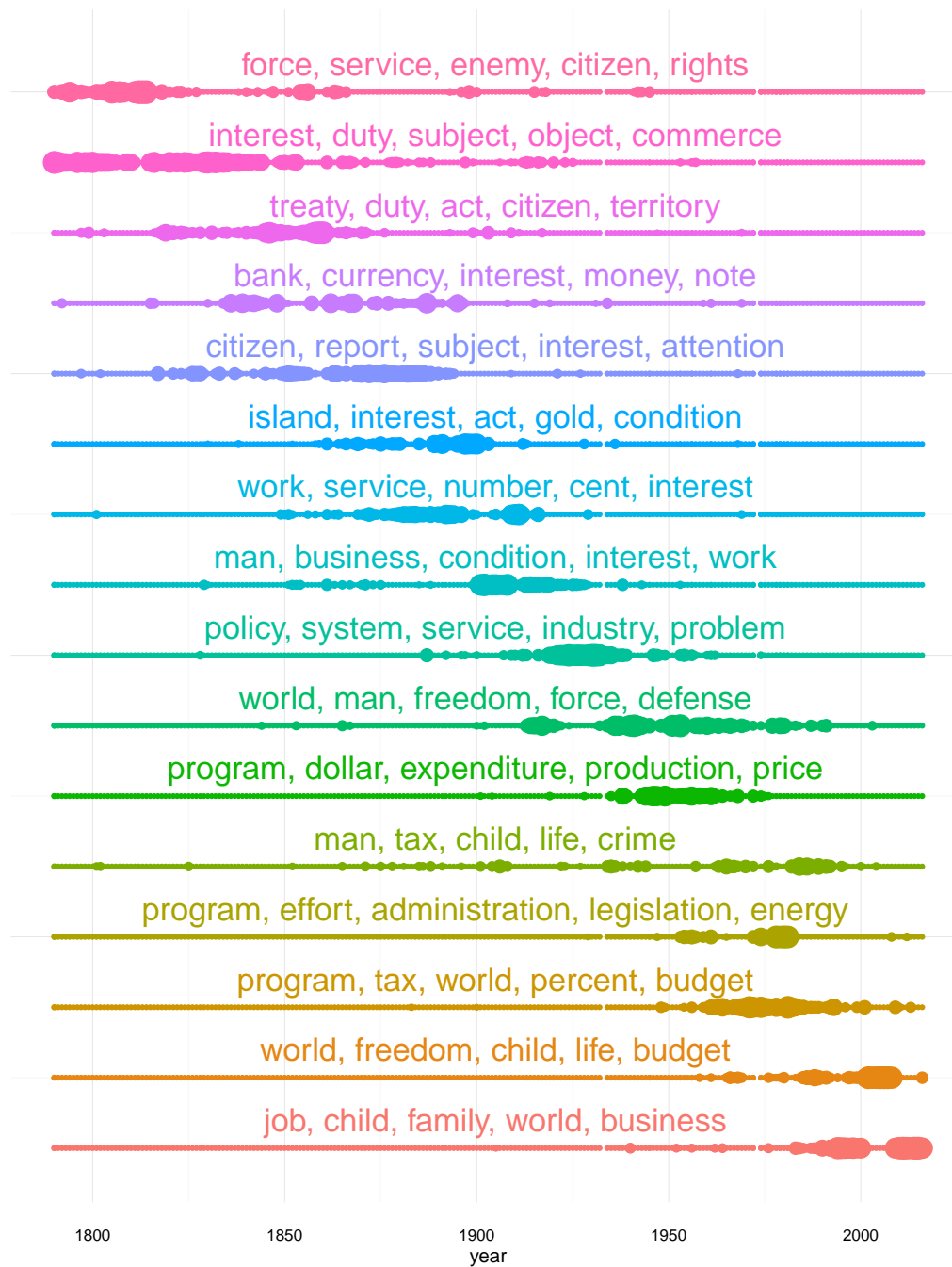


Figure 3: Distribution of topic model posterior probabilities over time on the State of the Union corpus. Top five words associated with each topic are displayed, with topics sorted by the median year of all documents placed into the respective topic using the maximum posterior probabilities.

```
> library(glmnet)
> model <- cv.glmnet(mat$tf[meta$train,], meta$y[meta$train], family = "binomial")
```

A boxplot of the predicted classes for each address is given in Figure 4. The algorithm does a very good job of separating the speeches. Looking at the odd years versus even years (the training and testing sets, respectively) indicates that the model has not been over-fit.

One benefit of the penalized linear regression model is that it is possible to interpret the coefficients in a meaningful way. Here are the non-zero elements of the regression vector, coded as whether the have a positive (more Obama) or negative (more Bush) sign:

```
> beta <- coef(model, s = model[["lambda"]][11])[-1]
> sprintf("%s (%d)", mat$vocab, sign(beta))[beta != 0]
[1] "job (1)"          "business (1)"     "family (1)"       "government (-1)"
```

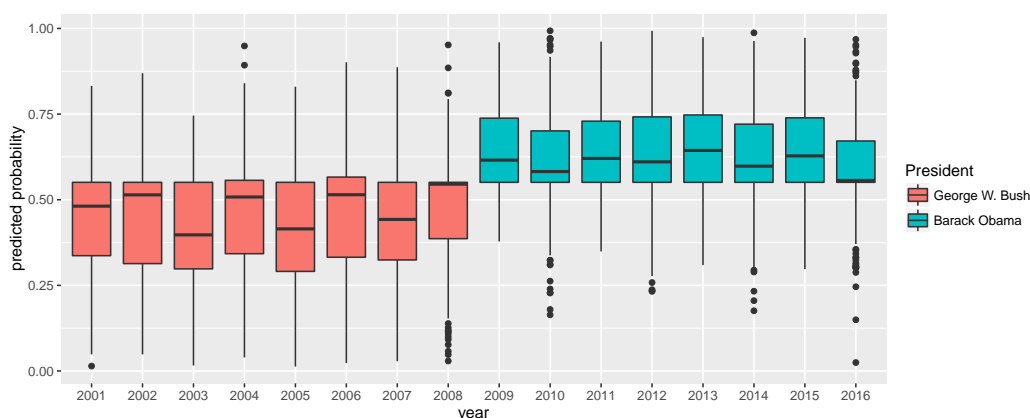



Figure 4: Boxplot of predicted probabilities, at the sentence level, for all 16 State of the Union addresses by Presidents George W. Bush and Barack Obama. The probability represents the extent to which the model believe the sentence was spoken by President Obama. Odd years were used for training and even years for testing. Cross-validation on the training set was used, with the one standard error rule, to set the lambda tuning parameter.

[5] "home (1)"	"citizen (-1)"	"terrorist (-1)"	"freedom (-1)"
[9] "education (1)"	"college (1)"	"weapon (-1)"	"deficit (1)"
[13] "company (1)"	"enemy (-1)"	"peace (-1)"	"terror (-1)"
[17] "hope (-1)"	"drug (-1)"	"kid (1)"	"regime (-1)"
[21] "crisis (1)"	"industry (1)"	"class (1)"	"income (-1)"
[25] "need (-1)"	"fact (1)"	"relief (-1)"	"bank (1)"
[29] "liberty (-1)"	"society (-1)"	"account (-1)"	"duty (-1)"
[33] "folk (1)"	"compassion (-1)"	"environment (-1)"	"inspector (-1)"

These generally seem as expected given the main policy topics of focus under each administration. During most of the Bush presidency, as mentioned before, the focus was on national security and foreign policy. Obama, on the other hand, inherited the recession of 2008 and was far more focused on the overall economic policy.

Conclusions

In this paper a basic description of the usage and philosophy behind the **cleanNLP** package has been presented. It is expected that some users will utilize the entirety of the underlying CoreNLP pipeline, internal R structures, and helper functions. Others may use only the corresponding data model to convert from a format such as CoNNL into a tabular data format more amenable to statistical analyses. In either extreme, or anywhere in between, the package provides powerful tools for applying exploratory, graphical, and model-based techniques to textual data sources.

Bibliography

- G. Angeli, M. J. Premkumar, and C. D. Manning. Leveraging linguistic structure for open domain information extraction. *Linguistics*, (24), 2015. [p9]
- T. Arnold and L. Tilton. *coreNLP: Wrappers Around Stanford CoreNLP Tools*, 2016. R package version 0.4-2. [p1]
- S. Buchholz and E. Marsi. Conll-x shared task on multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, pages 149–164. Association for Computational Linguistics, 2006. [p3]
- J. Chang. *lda: Collapsed Gibbs Sampling Methods for Topic Models*, 2015. URL <https://CRAN.R-project.org/package=lda>. R package version 1.4.2. [p1]
- D. Chen and C. D. Manning. A fast and accurate dependency parser using neural networks. In *EMNLP*, pages 740–750, 2014. [p5]

- E. F. Codd. *The relational model for database management: version 2*. Addison-Wesley Longman Publishing Co., Inc., 1990. [p3]
- M.-C. De Marneffe, T. Dozat, N. Silveira, K. Haverinen, F. Ginter, J. Nivre, and C. D. Manning. Universal stanford dependencies: A cross-linguistic typology. In *LREC*, volume 14, pages 4585–92, 2014. [p6]
- J. R. Finkel, T. Grenager, and C. Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 363–370. Association for Computational Linguistics, 2005. [p7]
- S. Green, M.-C. De Marneffe, J. Bauer, and C. D. Manning. Multiword expression identification with tree substitution grammars: A parsing tour de force with french. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 725–735. Association for Computational Linguistics, 2011. [p5]
- B. Grün and K. Hornik. topicmodels: An R package for fitting topic models. *Journal of Statistical Software*, 40(13):1–30, 2011. doi: 10.18637/jss.v040.i13. [p1]
- S. Hellmann, J. Lehmann, and S. Auer. Nif: An ontology-based and linked-data-aware nlp interchange format. *Working Draft*, 2012. [p3]
- K. Hornik. *NLP: Natural Language Processing Infrastructure*, 2016a. URL <https://CRAN.R-project.org/package=NLP>. R package version 0.1-9. [p1]
- K. Hornik. *openNLP: Apache OpenNLP Tools Interface*, 2016b. URL <https://CRAN.R-project.org/package=openNLP>. R package version 0.2-6. [p1]
- K. Hornik. *StanfordCoreNLP*, 2016c. URL <http://datacube.wu.ac.at/src/contrib/>. R package version 0.1-1. [p1]
- N. Ide and L. Romary. A common framework for syntactic annotation. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, pages 306–313. Association for Computational Linguistics, 2001. [p3]
- R. Jiang, R. E. Banchs, and H. Li. Evaluating and combining named entity recognition systems. *ACL 2016*, page 21, 2016. [p1]
- H. Lee, Y. Peirsman, A. Chang, N. Chambers, M. Surdeanu, and D. Jurafsky. Stanford’s multi-pass sieve coreference resolution system at the conll-2011 shared task. In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning: Shared Task*, pages 28–34. Association for Computational Linguistics, 2011. [p6]
- H. Lee, A. Chang, Y. Peirsman, N. Chambers, M. Surdeanu, and D. Jurafsky. Deterministic coreference resolution based on entity-centric, precision-ranked rules. *Computational Linguistics*, 39(4):885–916, 2013. [p6]
- C. D. Manning, M. Surdeanu, J. Bauer, J. R. Finkel, S. Bethard, and D. McClosky. The Stanford CoreNLP natural language processing toolkit. In *ACL (System Demonstrations)*, pages 55–60, 2014. [p1]
- D. Meyer, K. Hornik, and I. Feinerer. Text mining infrastructure in r. *Journal of statistical software*, 25(5): 1–54, 2008. [p1]
- G. Peters. State of the union addresses and messages. [p10]
- A. N. Rafferty and C. D. Manning. Parsing three german treebanks: Lexicalized and unlexicalized base-lines. In *Proceedings of the Workshop on Parsing German*, pages 40–46. Association for Computational Linguistics, 2008. [p5]
- K. Raghunathan, H. Lee, S. Rangarajan, N. Chambers, M. Surdeanu, D. Jurafsky, and C. Manning. A multi-pass sieve for coreference resolution. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 492–501. Association for Computational Linguistics, 2010. [p6]
- M. Recasens, M.-C. de Marneffe, and C. Potts. The life and death of discourse entities: Identifying singleton mentions. In *HLT-NAACL*, pages 627–633, 2013. [p6]
- R. Socher, A. Perelygin, J. Y. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*, volume 1631, page 1642. Citeseer, 2013. [p8]

- K. Toutanova and C. D. Manning. Enriching the knowledge sources used in a maximum entropy part-of-speech tagger. In *Proceedings of the 2000 Joint SIGDAT conference on Empirical methods in natural language processing and very large corpora: held in conjunction with the 38th Annual Meeting of the Association for Computational Linguistics-Volume 13*, pages 63–70. Association for Computational Linguistics, 2000. [p4]
- K. Toutanova, D. Klein, C. D. Manning, and Y. Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 173–180. Association for Computational Linguistics, 2003. [p4]
- H. Wickham et al. Tidy data. *Journal of Statistical Software*, 59(i10), 2014. [p1, 3]
- F. Wild. *lsa: Latent Semantic Analysis*, 2015. URL <https://CRAN.R-project.org/package=lsa>. R package version 0.73.1. [p1]

Taylor Arnold
Department of Mathematics and Computer Science
University of Richmond
Richmond, VA 23173 USA
tarnold2@richmond.edu