A Online Appendix: Additional Text Preparation Details

A.1 Conversion to Digital Format

Text gathered from a variety of different sources may not be immediately readable by a computer. First, the data itself might be pictures of text taken from an archive or hand-written manuscripts that are not yet digitized. In these cases, Optical Character Recognition (OCR) technologies may be required.[1] Even if they have been digitized, texts may be stored digitally using different encodings and therefore not immediately usable as one corpus.

A.2 Bag of words

The bag-of-words assumption underlies many statistical models due to its relative simplicity. The simplest method of expanding beyond this approach is to allow each item in the dictionary to represent a multi-word phrase rather than a single word. Because the space of ordered word pairs (bigrams) and ordered word triples (trigrams) is significantly higher than using a single word, this procedure is often coupled with some method of selecting the most relevant phrases (Jensen et al., 2012; Gentzkow and Shapiro, 2010). This process can often engender subtle difficulties in interpretation (see for example the critique of trigrams in Spirling (2012)). More complex approaches have included the use of string kernels (Spirling, 2011) and word transition distributions (Wallach, 2006). We choose to focus here on the simple bag-of-words model but emphasize that vocabulary can be manually modified to include contextually appropriate phrases.

A.3 Additional pre-processing options

As is typically the case in text analysis, the illustrations presented in this paper required a good deal of preprocessing. In determining how to preprocess our data for each of these examples, we drew on substantive knowledge of the relevant language, cases, and questions, and suggest that others do the same. But because no analysis is the same, researchers may need to preprocess their data in ways not presented in this paper. For example, users may wish to correct spelling errors, especially when analyzing survey responses or social media posts where such errors are common. This is easily accomplished in `R` with the built-in `aspell()` function. Users may also wish to expand contractions (if they are not stemming), to remove punctuation, to make text lowercase (except for

[1] OCR works by using pattern recognition to identify patterns within the image file that look like characters and transfer them into machine-encoded text. Some OCR software are dictionary-based, searching for word within the dictionary that looks most like the image. While OCR software is rarely, if ever, 100% accurate, clearly-written texts can often achieve accuracy rates of above 90%, which is enough to understand the content of the text and to use automated content analysis. Even with high error rates, errors are unlikely to be correlated with most quantities of interest, so the measurement error can be corrected for. There are many open-source OCR software packages, (see for example FreeOCR: `http://www.free-ocr.com/` or Tesseract: `https://code.google.com/p/tesseract-ocr/`) and we recommend that users try out several packages on a given text, as none are perfect, but some may work better with particular image files. If OCR is not possible it is often possible to pay for data entry.

proper nouns), etc., most of which can be accomplished easily with `tm` or `txtorg`.

B Online Appendix: A Review for Text Analysis Techniques for Comparativists

In this online appendix, we provide a brief pedagogical overview of supervised and unsupervised methods with an emphasis on choosing the best approach for applied research. We refer interested readers to Grimmer and Stewart (2013) for more detailed discussion on particular models, but here we focus more on examples to help applied comparative politics researchers. We emphasize two aspects of analysis throughout: the processes by which the analyst incorporates information into the model and the importance of validation, by which we mean checking model results with substantive knowledge and reference to the original text.

Oftentimes the virtues and limitations of methods are best illustrated through a familiar set of documents. In the next few sections we will illustrate methods with toy examples using a corpus comprised of the last 6 decades of research articles from the *American Political Science Review* containing the phrase "comparative politics". We obtain the word count information from JSTOR's Data-For-Research site, selecting articles from 1950-2012. We limit the classification to research articles and choose only articles in excess of 5 pages (in an attempt to further remove reviews, rejoinders etc.), yielding 417 articles. These texts are accompanied by a variety of metadata elements including the authors, titles and year of publication. At each stage we point the reader to applied examples of the methods we discuss.

B.1 Supervised Methods

In their most basic form, supervised methods are a way of replicating work done by the human analyst on a small scale, to a much larger set of documents. This allows the analyst to undertake an analysis that one could imagine a human performing with infinite time but would for all practical purposes be intractable. We start with simple word counting approaches and their natural extension, document scaling by weighted word counts.

Keywords Methods and Document Scaling The simplest form of supervision is counting human-selected keywords. The choice to focus on the counts of particular chosen words is itself a case of a very simple supervised model. When the quantity of interest is precisely represented, keywords can be a conceptually and computationally simple approach. Some approaches (such as Yoshikoder)[2] assist the user in interpreting the text by placing keywords within "context" (showing example sentences and phrases which contain the keyword). These methods are a helpful validation method but they still assume that the use of a particular word is the same across different documents regardless of the context in which it is used.

Figure 1 shows a simple keyword trend tracking the use of the word "causal" in *APSR*. This accords with our understanding of a general rise in interest in causal inference methods over the last 6 decades. While informative, it is important to emphasize that

[2] `http://conjugateprior.org/software/`

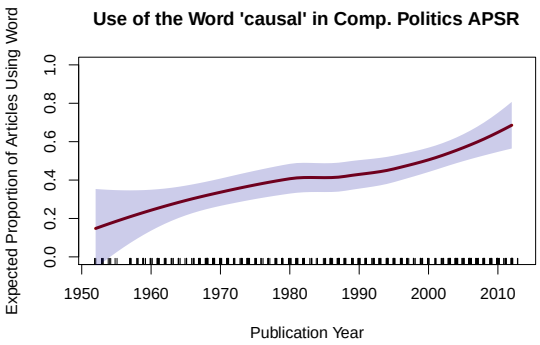

Figure 1: Illustration of Simple Keyword trends. This shows the rise in the proportion of articles in *APSR* using the word 'causal.' This is a suggestive trend which accords with our understanding of the literature but requires additional supervision from the researcher to determine how much we can infer. The plot shows the Loess-smoothed estimate of the expected proportion of articles using the word "causal" with 95% confidence intervals.

this need not indicate that researchers are using causal inference methods or even that the use of the word is consistent over the time frame. For that we would need to turn to more complex methods.

A common extension to counting methods is to consider weighted counts. These methods are most often applied to sentiment analysis where dictionaries encode the valence of individual words, for example, "angry" might get a more negative sentiment score than "annoyed." There are dictionaries available for affect, cognitive mechanisms, sentiment and various topics (Stone et al., 1966; Pennebaker et al., 2001). However, these dictionaries were developed for particular types of texts and may not extend well to other domains (Grimmer and Stewart, 2013).

Dictionaries of weighted terms can also be learned automatically from human annotated task. In this setting, the user assigns texts to (generally extreme ends of) categories and uses a model to determine words that are most "distinctive" or predictive of that class. Words that have high use under one class but low use under another are given higher scores. Care must be taken with the interpretation of continuous scores generated from these methods. The continuous nature is (generally) derived from the predictive power of the features under a discrete classification system. This need not always correspond to an intuitive notion of intensity (e.g. "more conservative") although it does work well in many cases.[3]

To emphasize this point we give an example from the *APSR* corpus. Here we leverage the authorship metadata to group documents into those which are co-authored and those which are solo-authored. In order to characterize what is distinctive about the co-authored articles we create a "co-author" score where we scale higher scores that indicate that the

[3]Monroe et al. (2008) provide a comparison of various weighting methods of words for the purposes of feature selection.

words are more predictive of co-authorship.[4] We give the top 10 words for each side in Table 1. The results suggest that quantitative empirical work is often co-authored

	Solo Words	Score	Coauthor Words	Score
1	rearmament	-5.061	ethnopolitical	5.259
2	lebanese	-4.846	elf	4.212
3	theology	-4.788	civics	4.208
4	machiavelli	-4.635	humphrey	4.078
5	crops	-4.514	contemporaneously	3.952
6	loc	-4.444	rd	3.95
7	hitler	-4.444	AY	3.866
8	colonization	-4.444	pie	3.862
9	cdu	-4.444	supervisors	3.845
10	mussolini	-4.42	admissible	3.786

Table 1: Top words indicating coauthored or solo-authored work with scaled values. Words with higher absolute value scores have a stronger connection.

whereas historical and qualitative work is more likely to be solo-authored. The words don't necessarily capture a theoretical conception of "collaborativeness"; they are simply the most predictive of co-authorship.

This example affords us the opportunity to emphasize the importance of validation in computer-assisted text methods. For readers of comparative politics it may be easy to look at the set of coauthor words and tell a consistent story which implies a particular contextual meaning for each word; "elf" refers to the ethno-linguistic fractionalization index, "rd" refers to regression discontinuity estimators etc. "Humphrey" the reader may naturally assume refers to Macartan Humphreys, as his research is often quantitative and is often coauthored. However, examination of the documents in our sample which contain the word "humphrey" reveals that the finding is driven largely by a few outliers. One article which uses "humphrey" an astonishing 55 times is titled "Continuity and Change in American Politics: Parties and Issues in the 1968 Election" (Converse et al., 1969). The article discusses Vice President Hubert Humphrey, a subject which clearly has a quite different interpretation. This example illustrates the need for checking model interpretation against the original texts even when a collection of words appear to tell a very clear story.

Example Applications of Keyword and Scaling Methods Several good examples of keyword methods can already be found in the comparative politics literature. Johnston and Stockmann (2007) use stories from the *China Daily* to study Chinese attitudes toward Americans immediately after the 2004 tsunami in South Asia. They identify positive and negative terms surrounding references to the United States in order to measure attitudes toward American relief efforts following the tsunami. Stockmann (2011) analyzes how media marketization in China influences views toward the United States. She finds that after extensive media marketization in the early 2000s, Chinese newspapers use more negative words surrounding the United States than positive.

[4]Specifically we use Taddy (2013)'s inverse regression procedure which fits a regularized multinomial logistic regression with words as the output and the co-authorship indicator as the predictor. Scores are essentially the regularized log-odds of word use.

Scaling is the most widely used automated text analysis within comparative politics due to its prominent role in the study of partisan ideology. Perhaps the most prominent example is the WordScores method (Laver et al., 2003) for analyzing ideology in the Comparative Manifestos Project. As this literature is quite well developed in comparative politics we do not dwell on it at length. For more methodological details on the connections between scaling methods and supervised learning we refer readers to Lowe (2008); Beauchamp (Beauchamp); Taddy (2013).

Classification/Prediction Document classification is perhaps the most common form of supervised learning. In this setting researchers develop a categorization by hand and use it to label a random subset of the documents which we call the 'training set.' The model learns a set of parameters from the training set which it uses to assign the remaining documents into our original categories. Classification is most useful in cases where (1) discrete individual decisions need to be made or (2) where an extensive coding system already exists.[5] There are an enormous set of possible classification algorithms that the analyst can employ, but their common structure makes it possible to provide general advice that applies to nearly the entire set.

In order to provide some context, we outline an example workflow for document classification. The researcher must first develop a categorization scheme which assign each document to a category in a way that is both mutually exclusive and exhaustive. Best practice is to develop a codebook which provides sufficient detail that an informed third party would be able to assign a new document within the existing scheme (Krippendorff, 2012). The researcher would ideally want all of the documents hand-coded into this categorization scheme, but with a large number documents, hand-coding each individual document would be too time-consuming and expensive. Instead, we hand code only a sample, with the algorithm classifying the remaining unseen documents.

To teach the algorithm how to classify, researchers select a (preferably random) sample of documents to be manually coded by human coders into the categories. In order to ensure that the categories are consistent across coders, each document is usually coded by two or more researchers (who are provided with the aforementioned codebook) with a final arbiter comparing the coding to measure the "inter-coder reliability", or how consistent the coding is between researchers. In practice this is often an iterative process of developing the coding scheme and refining the codebook. Once the coders have achieved a high inter-coder reliability, the resulting "training set" of coded documents and the remainder of the uncoded documents are then given to the classifier as a term-document matrix. Using this representation the algorithm learns the parameters for a model which can classify unseen documents. This classifier is then applied to the remainder of the corpus.

A significant advantage of supervised document classification is the relative ease in evaluating machine performance. The accuracy of the classifier can be checked by manually inspecting new documents and comparing the classifier's coding to manual cod-

[5]The canonical example of the first case is spam filtering, where your email system must decide for each email if it is spam or regular. The Congressional Bills Project which classifies documents according to the topic system developed by the Policy Agendas Project is an example of the second case (Hillard et al., 2008).

ing. With a sufficiently large training set classifier accuracy can be assessed using cross-validation (Grimmer and Stewart, 2013).

Although it is relatively easy to assess individual document codings, it is often beneficial, although significantly harder, to assess the calibration of the model's predictions. For example imagine a setting where we classify emails into 'spam' or 'not spam.' Assessing whether an individual item marked 'spam' is correctly identified is often trivially easy. It takes significantly more data to assess whether items marked as 70% likely to be spam are actually spam 70% of the time in expectation. While many classification algorithms tend to make strong individual predictions, these predicted probabilities are often overly confident.

Calibration of the model becomes particularly important when we are interested in making inference about a group of documents. If, for example, we wanted to assess what proportion of *APSR* articles in our corpus are about 'institutions' we could train a classifier and then sum up the predicted probability that each document falls into the 'institutions' category. However, if our classifier is poorly calibrated, this estimate of the group proportions can be severely biased.

One method developed within political science, `ReadMe`, leverages this particular goal to provide more accurate estimates of category proportions than classifiers focused on individual classifications (Hopkins and King, 2010). `ReadMe` returns only the group category proportions and does not provide individual document level categorizations. This focus on estimating the population proportion correctly allows for increased accuracy but makes it inappropriate for settings where the classification of each individual document must be known.

The workflow for using `ReadMe` is substantially similar to individual document classification. The lack of individual document assignments can mildly complicate the process of validation, but ultimately it is no more difficult than attempting to assess the calibration of classification probabilities in the individual classifier setting. The entire workflow for using `ReadMe` is summarized in the user manual for the software (Hopkins et al., 2010) with much of the advice also being applicable to the broader set of classification algorithms.

Example Applications of Document Classification King et al. (2013) provide one recent example of using `ReadMe` in comparative politics. King et al. (2013) download millions of blogposts before the Chinese government is able to censor them, then return to the blog posts later to see whether they were censored, providing one of the first large-scale measurements of government censorship in China. In one part of the paper, the authors use `ReadMe` to test whether censorship in China is focused on removing blog posts about collective action events or removing blog posts with criticism of the state. Jamal et al. (nd) provide another example. They examine views of America that are being expressed in Arabic and on Twitter. Rather than relying on public opinion data to understand views of America, Jamal et al. (nd) innovate by analyzing millions of tweets about America. They also create specific categories to analyze responses to events, like the Boston Marathon bombing.

When and How to Use It Supervised methods are best used when the analyst is looking to identify a particular quantity within the text. In the case of document classi-

fication errors within the algorithm can be straightforwardly assessed by comparing the algorithm's decisions to the human-created codebook.[6] For keyword methods using pre-constructed dictionaries of words be sure to validate the resulting methods to insure that they represent the intended concept.

The software tools for document scaling and classification are particularly well developed. Within the `R` language, there are numerous packages including `RTextTools` (Jurka et al., 2013) for classification and `austin` for scaling (Lowe, 2013).[7]

B.2 Unsupervised Methods

Where supervised methods amplify human effort by attempting to replicate human effort expended at the beginning of the process, unsupervised methods suggest new ways of organizing texts. This in turn means that human effort is primarily focused on the interpretation of the results. Thus these methods are useful when seeking to broadly characterize what a corpus of texts is about without strong *a priori* assumptions about what that might mean. While originally used primarily for exploratory analysis, increasing attention has been paid to using these methods in the context of measurement, generally in contexts where the development of a coding system would be prohibitively expensive (Quinn et al., 2010; Grimmer, 2010) .

Keywords and Scaling As in the supervised case we can extend the word frequency approach to weighted word frequencies. We posit a continuous latent variable which explains the text such that each document is assigned to lie along a continuum. This is the approach taken in the Wordfish algorithm for studying political ideology (Slapin and Proksch, 2008) and is the unsupervised analog of our analysis with the coauthorship variable used previously. [8]

Document Clustering Document clustering assumes that each document belongs to a single latent group, and this group provides the best explanation for word use. This approach has been used in political science for analyzing the contents of Senate press release (Grimmer, 2010) and speech on the floor of the House (Quinn et al., 2010). Clusters of documents can either each be distinct or nested into hierarchies themselves.

For longer and more complex documents (e.g. our *APSR* corpus) the assumption that each document falls within only a single cluster is too restrictive. Mixed-membership models, such as Latent Dirichlet Allocation (LDA) (Blei, 2012), assume that each word comes from a single topic, and that each document comes from a mixture of topics. Thus, a document is represented as the proportion of its words that come from each topic.[9]

[6]Specifically we often validate accuracy via cross-validation. See Grimmer and Stewart (2013) for details.

[7]Documentation on `RTextTools` can be found at `http://www.rtexttools.com/`. Documentation for `austin` can be found urlhttp://conjugateprior.org/software/austin/. Both have an excellent set of examples for getting started.

[8]Of course there is no guarantee that the recovered dimension will correspond to any specific political concept (Grimmer and Stewart, 2013). In general, unsupervised scaling methods will characterize the dominant source of variation in the texts whether that variation is topical, ideological or stylistic. For other examples of unsupervised scaling methods, see Elff (2013).

[9]To help distinguish the two types of models we use the term "cluster" when discussing single mem-

For our APSR corpus we run an LDA model with six topics and in Table 2 display seven informative words which characterize each topic.

1	press,democratic,international,democracy,countries,variables,institutions
2	elections,election,local,members,models,population,levels
3	military,war,conflict,civil,people,leaders,empirical
4	groups,group,interest,problems,foreign,law,latin
5	behavior,cases,society,soviet,values,issue,approach
6	party,parties,model,electoral,vote,voting,effects

Table 2: Top words for a 6 topic LDA model on the APSR corpus.

We emphasize that these topics are discovered from the texts using the patterns of word co-occurrences and are not assumed by the researcher. To give a concrete example, a document mostly coming from topic 2 (about elections and local politics) is "French Local Politics: A Statistical Examination of Grass Roots Consensus" (Kesselman, 1966) and it also draws from the 5th topic on behavior and the 6th topic on parties and voting.

When and How to Use Unsupervised Text Models Unsupervised learning is best applied when the analyst is interested in the contents of a corpus but do not have strong expectations for its structure. Generally all that is necessary to perform document clustering or topic modeling is the algorithm of choice and an assumption about the number of latent groups to be estimated. There is no "correct" specification in a general sense, but different approaches will provide different types of structure. For example specifying the number of clusters or topics provides views of the data at different levels of granularity.

Unsupervised scaling using the Wordfish algorithm (Slapin and Proksch, 2008) is available in the `austin` package (Lowe, 2013). The basic LDA model is implemented in several R packages including `lda` (Chang, 2012) and `topicmodels` (Grün and Hornik, 2011).

Choosing a Strategy When choosing a method for automated text analysis the key is to focus on the best use of human effort. At an abstract level the distinction between unsupervised and supervised learning is extremely clear: supervised learning is learning by human-provided example whereas in unsupervised learning those examples are discovered from the data. In practice the line between the two approaches is somewhat hazier. Nearly all good coding schemes are developed by a combination of *a priori* theory and iterative inspection of the data. Thus even in supervised models, there is a "concept discovery" process. With supervised learning this comes through model fitting and checking errors, in unsupervised learning the concept discovery is simply part of the model.

How then should we think about the difference between the two approaches in applied work? Supervised learning offers a greater degree of control: concepts are completely and exclusively enumerated. This is a powerful assumption which tends to make models easier to estimate and results easier to check (because the concept of an error is very well defined). When the space of possible concepts is easy to define this is considerably more tractable.

bership models and "topic" when describing "mixed membership models."

The choice between supervised and unsupervised learning is subtle in the general case but often is quite obvious in practice. No matter what you choose it is important to validate your results. This helps to prevent conceptual slippage between the statistical model of the text and the theoretical concept which we are claiming to measure. For approaches to validation see Grimmer and Stewart (2013); Quinn et al. (2010); Lowe and Benoit (2013).

C Online Appendix: Machine Translation via translateR

As discussed in the paper great strides have been made in machine translation researcher, especially by large corporations like Google and Microsoft. This paper makes use of a new `R` package `translateR` to help researchers translate multilingual corpora into a common language. `translateR` provides bindings for the Google and Microsoft Translation APIs and can export translated text in a format that easily works with the `stm` package or other text analysis packages.[10] `translateR` has one workhorse function, `translate()`, through which it provides access to both supported APIs. `translate()` takes text in two formats. First, users can provide as input a dataframe where one column is text and the remaining are metadata. In this case, `translate()` returns the original dataframe with the translated text bound to the original data set. This is important for use with `stm`, as the metadata can then be used as covariate information to estimate a structural topic model for the translated text. Second, users can provide text in the form of a character vector. This is valuable for users translating a document-term matrix. The column names of a document-term matrix (i.e., the terms) can be passed to `translate()` as a vector. The translated vector can then be used to rename the columns, thus "translating" the DTM from one language into another.

`translateR` also provides a number of "sanity checks" to protect users. For instance, `translateR` uses the `textcat` package (Hornik et al., 2012) to confirm that users have selected the proper source language, a functionality that is very important, given that simple typos in language can could cost the user money when used with large corpora. Finally, `translateR` parallelizes the translation process, thus reducing the runtime necessary to translate an entire corpus. In sum, `translateR` provides a powerful mechanism for `R` users to access cutting edge machine translation tools and in a way that comports with topic modeling software.[11]

C.1 Usage

To use the Google API, users must provide a valid API key, available at `https://developers.google.com/translate/v2/getting_started`. If a Google API key is passed

[10] Discussion of the relevant advantages of each, including access cost, is beyond the scope of this document. Though we note that as of summer 2014, the Microsoft API is that it will accept longer documents.

[11] An alternative, less developed, `R` package for translation is the `translate` package which accesses the Google API only and can only process one document at a time, without any of the additional features discussed above.

to `translate()`, the Google API will be used to translate the text. To use the Microsoft API, users must pass a valid client ID and client "secret value," available at `http://msdn.microsoft.com/en-us/library/hh454950.aspx`. Note that users must pass to `translate()` either a Google key or both a Microsoft client ID and a Microsoft client secret value.

The `translateR` packages installs with a small dataset constructed from the Enron corpus (Klimt and Yang, 2004), a corpus constructed from emails sent during the Enron scandal and in the public domain. The small subset has ten rows and three columns, two of which are metadata (the email subject and date). The text is simply the body of the email. The column names are "email," "date," and "subject." In the following example, we demonstrate different ways in which the software can be used to translate the original text into German. Package help files contain similar documentation.

```
# Load example data. Three columns, the text content ('email')
# and two metadata fields ('date' and 'subject')
data(enron)

# Google, translate column in dataset
google.dataset.out <- translate(dataset = enron, content.field = 'email',
                          google.api.key = my.api.key,
                          source.lang = 'en', target.lang = 'de')

# Google, translate vector
google.vector.out <- translate(content.vec = enron$email,
                          google.api.key = my.api.key,
                          source.lang = 'en', target.lang = 'de')

# Microsoft, translate column in dataset
google.dataset.out <- translate(dataset = enron, content.field = 'email',
                          microsoft.client.id = my.client.id,
                          microsoft.client.secret = my.client.secret,
                          source.lang = 'en', target.lang = 'de')

# Microsoft, translate vector
google.vector.out <- translate(content.vec = enron$email,
                         microsoft.client.id = my.client.id,
                         microsoft.client.secret = my.client.secret,
                         source.lang = 'en',
                         target.lang = 'de')
```

The full set of arguments for `translate()` is as follows, though we refer users to `translateR` package documentation for details.

- **dataset:** A dataframe with a column containing the text to be translated.

- **content.field:** If a dataframe is passed to "dataset", the name of the column containing the text must be passed to "content.field".
- **content.vec:** A character vector of text. This is an alternative to "dataset"/"content.field".
- **google.api.key:** To use the Google API, an API key must be provided. For more information on getting your key, see `https://developers.google.com/translate/v2/getting_started`.
- **microsoft.client.id:** To use the Microsoft API, a client id and a client secret value must be provided. For more information on getting these, see `http://msdn.microsoft.com/en-us/library/hh454950.aspx`. NOTE: you do not need to obtain an access token. translateR will retrieve a token internally.
- **microsoft.client.secret:** To use the Microsoft API, a client id and a client secret value must be provided. For more information on getting these, see `http://msdn.microsoft.com/en-us/library/hh454950.aspx`. The client secret value is a unique identifying string obtained when registering with Microsoft (see the link for more information). NOTE: you do not need to obtain an access token. translateR will retrieve a token internally.
- **source.lang:** The language code that corresponds with the language in which the source text is written. The translators use different language codes, so select accordingly. To see a list of language codes, enter getGoogleLanguages() or getMicrosoftLanguages() for Google or Microsoft, respectively.
- **target.lang:** The language code that corresponds with the language into which the source text is to be translated. The translators use different language codes, so select accordingly. To see a list of language codes, enter getGoogleLanguages() or getMicrosoftLanguages() for Google or Microsoft, respectively.

Note that the `source.lang` and `target.lang` arguments require language codes specific to the Google and Microsoft APIs. To make it easier for users to access these codes, we provide two functions. `getGoogleLanguages()` and `getMicrosoftLanguages()` print the full listing of languages supported by each API and their corresponding code so that users do not have to look this information up online.

D Online Appendix: Core STM Technical Details

We briefly state the data generating process for the STM, referring readers elsewhere for additional details (Roberts et al., 2014, nd, 2013). The generative process for each document (indexed by d) can be summarized as:

1. Draw the document-level attention to each topic from a logistic-normal generalized linear model based on document prevalence covariates X_d.
   $\vec{\theta}_d | X_d\gamma, \Sigma \sim \text{LogisticNormal}(\mu = X_d\gamma, \Sigma)$

2. Form the document-specific distribution over words representing each topic (k) using the baseline word distribution (m), the topic specific deviation κ_k, the content covariate group deviation κ_g and the interaction between the two κ_i.
   $\beta_{d,k} \propto \exp(m + \kappa_k + \kappa_{g_d} + \kappa_{i=(kg_d)})$

3. For each word in the document, $(n \in 1, \ldots, N_d)$:

   - Draw word's topic assignment based on the document-specific distribution over topics.
     $z_{d,n}|\vec{\theta}_d \sim \text{Multinomial}(\vec{\theta})$
   - Conditional on the topic chosen, draw an observed word from that topic.
     $w_{d,n}|z_{d,n}, \beta d, k = z \sim \text{Multinomial}(\beta_{d,k=z})$

The model is completed with regularizing prior distributions for γ, κ and Σ. Both prevalence and content covariates are optional, When content covariates are not included the distribution over β can optionally be optimized directly as in the standard LDA model. Without prevalence or content covariates, the model reduces to a logistic-normal topic model (Blei and Lafferty, 2007) rather than LDA as the Logistic Normal allows the topics to be correlated.

We estimate the model using a variational EM algorithm as implemented in the `stm` package in `R`. All reported covariate effects are estimated in the package as well using a method of composition style approach to dealing with measurement uncertainty in the latent topics (Roberts et al., 2014).

E Online Appendix: Topic Correlation Graph Estimation

Here we introduce and discuss two estimation procedures we provide for producing topic correlation plots both of which are implemented in the `stm` package. The first method is conceptually simpler and involves a simple thresholding procedure on the estimated marginal topic covariance matrix and requires a human specified threshold. The second method draws on recent literature undirected graphical model estimation and can be automatically tuned. For Figure 2 in the paper we use simple thresholding measure.

Simple Thresholding Taking the correlation of the MAP estimates for the topic proportions θ yields the marginal correlation of the mode of the variational distribution. Then we simply set to 0 those edges where the correlation falls below the user threshold. A method of composition approach can be used to integrate over the uncertainty in the topic means by repeatedly drawing from the variational posterior for θ and calculating the correlation. This makes it possible to derive confidence intervals on the topic correlations.

Graph Estimation An alternative strategy is to treat the problem as the recovery of edges in a high-dimensional undirected graphical model. In these settings we assume that observations come from a multivariate normal distribution with a sparse precision matrix. The goal is to infer which elements of the precision matrix are non-zero corresponding to edges in a graph. In an influential piece, Meinshausen and Bühlmann (2006) showed

that using sparse regression methods like the LASSO it is possible to consistently identify edges even in very high dimensional settings. Drawing on this work Blei and Lafferty (2007) use the Meinshausen and Bühlmann (2006) approach to estimate a topic graph in the Correlated Topic Model by running a LASSO regression on the variational means. The variational means are plausibly multivariate normal but are not the true quantity of interest.

We use the recently developed Nonparanormal SKEPTIC procedure which was developed to extend previous results to the non-Gaussian case (Liu et al., 2012; Zhao et al., 2012). Essentially the data are transformed using a Gaussian Copula approach and then tested using a LASSO as in Meinshausen and Bühlmann (2006). Thus we are able to run model selection on the true quantity of interest: the MAP estimates for θ. Model selection for the scale of the L_1 penalty is performed using the rotation information criterion (RIC) which estimates the optimal degree of regularization by random rotations. The authors note that this selection approach has strong empirical performance but is sensitive to under-selection of edges (Zhao et al., 2012). We choose this metric as the default approach to model selection to reflect social scientists' historically greater concern for false positive rates as opposed to false negative rates.

The Nonparanormal SKEPTIC procedure has been shown to have excellent theoretical properties and empirical success in large genomics data sets (Liu et al., 2012). In high-dimensions (a large number of topics in our case) the procedure can be shown to have optimal parametric convergence rates even when the data are truly Gaussian. The procedure also has the advantage of identifying significant negative relationships between topics and effectively visualizing those for the user is a matter of future research.

We note that in models with low numbers of topics the simple procedure and the more complex procedure will often yield identical results. However, the advantage of the Nonparanormal SKEPTIC procedure hat we outline here is that it scales gracefully to models with hundreds or even thousands of topics - specifically the set of cases where some higher level structure like a correlation graph would be the most useful.

F Online Appendix: Additional Information on Empirical Examples

F.1 Fatwah Analysis

The Fatwah texts come as raw Arabic texts, so preprocessing is necessary. We first clean the text of punctuation marks and stray characters, and standardize some variants of Arabic letters that are functionally equivalent. We then remove numbers, reflecting our belief that these are not important for differentiating topics. We also believe that stop words are not important for our goal of uncovering the topics discussed by Muslim clerics so we remove stop words according to a custom list we developed in Arabic. We then remove prefixes and suffixes from remaining words using the "light 10" Arabic stemmer (Larkey et al., 2007) with minor modifications and implemented through the python extensibility available in txtorg. Stemming is appropriate for this application because we expect topics to be primarily related to the core concepts conveyed by each stem,

rather than specific prefixing, infixing, or suffixing. After preprocessing the Arabic text as described, we split the corpus into tokens by dividing the text at spaces in the text and removing any remaining whitespace. From these tokens, we construct a document-term matrix that can be read into stm.

F.2 Snowden Analysis

In this section, we provide additional detail concerning the Snowden analysis, including preprocessing, estimation, and interpretation of additional topics not discussed in Section 4.2, and also alignment across the two translation approaches.

F.3 Preprocessing

In the paper we analyze unique tweets, leaving the analysis of a combined retweet/original tweet corpus to future work which could study the intensity of particular messages. For both the Arabic and Chinese, we removed exact duplicates. However, some retweets are not exact matches, for any number of reasons. To address this issue in the Arabic corpus, we also removed tweets containing the string "RT", which signifies a retweet. For the Chinese corpus, we removed text containing the phrase, zhuanfa weibo, which is used to identify retweets, translating "forwarded Weibo."

To preprocess the Chinese text for full text translation, after translating the text, we stemmed the corpus and removed stopwords and infrequent words. For the document-term matrix translation, we first segmented the chinese corpus using the Stanford Word Segmenter (Chang et al., 2008; Tseng et al., 2005), though we also could have accomplished this with `txtorg`. Then, with both the Chinese and Arabic corpora, we converted the remaining terms to lowercase, stemmed them, removed stopwords, and removed all terms less than two characters in length. Following this procedure, we created a document-term matrix from the segmented corpus, then translated the vocabulary and combined any columns (terms) that translated into the same English term. We removed infrequent terms and then conducted the analysis on the remaining.

F.4 Alignment of Asylum, Human Rights, and Attack topics

To examine the alignment between topics estimated with full text translation versus document-term matrix translation, we marginalize over the betas, convert the vocabulary to lower case, and merge the betas by vocabulary. We then calculate the correlation across the two translation methods and generate the correlation plot displayed in Figure2.

The "Human Rights" topic in full text translation is that with the frequent words "american," "countri," and "peopl," which matches to the "Human Rights" topic in the DTM translation with frequent words "american," "country," and "traitor." This match can be seen in row 3, column 5.

The "Attack" topic in full text translation is that with the frequent words "china," "newspap," and "snowden," which matches to the "Attack" topic in the DTM translation with frequent words "china," "american," and "chinese." This match can be seen in row 15, column 1.

The "Asylum" topic in full text translation is that with the frequent words "asylum," "ecuador," and "snowden," which matches to the "Asylum" topic in the DTM translation

with frequent words "ecuador," "shelter," and "political." This match can be seen in row 12, column 10.

As evident in the figure, each pair matches well with its partner but the topics of interest are also sometimes correlated with a third topic. Thus, they appear to be identifying a common underlying topic, though we caution readers from drawing too strong a conclusion, as they are not (and cannot be expected to be) perfectly correlated.

F.4.1 Additional topics

Figure 3 displays the topics and the expected topic proportions for the full text translation, while Figure 4 does the same for the DTM translation. In Figure 3, the "Human Rights," "Attack," and "Asylum" topics (discussed at length in Section 4.2) are the 7^{th}, 12^{th}, and 5^{th}, respectively (counting down from the top). In the Figure 4, they are the 1^{st}, 8^{th}, and 5^{th}, respectively.

For the sake of brevity, we discussed only those topics in the body of the paper. However, other topics are potentially interesting. For instance, in the text translation, we observe a topic about signing a petition to stand with Snowden and stop the fighting. The "Petition" topic is associated with terms like "snowden", "spi", "share", "edward", "accus", "stop", and "pheoniz". One related tweet on Weibo reads (after translation), "Stand with Edward Snowden # #Snowden! BarackObama - Stopped spying #NSA. Signed the petition and share." It has an expected topic prevalence of 6.22%.

G Online Appendix: Benchmarking common pre-processing tools

As the total volume of text increases, the ability to efficiently manage text at scale becomes important. We argue here that once researchers start to work with larger volumes of text, an index-based management system like that provided by `txtorg` quickly becomes essential. An index is a data structure that allows the user to search the full corpus without without scanning every document, allowing for fast search operations even with millions of documents. The Lucene-powered index is essentially represented as a file containing a list of all the terms in the corpus, along with a link to the specific documents in which that term appears. Thus, instead of scanning every document in search of a particular term, `txtorg` scans the list of all the unique terms in the index, locates the match, and returns the documents linked to that term. In addition to searches through the corpora, `txtorg` also permits fast searches of metadata and joint searches of both metadata and text. `txtorg` can handle complex queries, including regular expressions, booleans, and a host of Lucene-supported options.

Once `txtorg` reads in a corpus, a new Lucene index is created and the original corpus may be discarded or stored elsewhere. This new index is considerably smaller than the original data permitting local storage of what might otherwise be an unwieldy corpus.[12] From the Lucene index, a user can export a DTM even if the original documents are not available.

[12]For example, we index an Arabic corpus of size 195.3 MB, which yields an index of size 78.1 MB. The relationship between corpus size and index size is approximately logarithmic rather than linear.

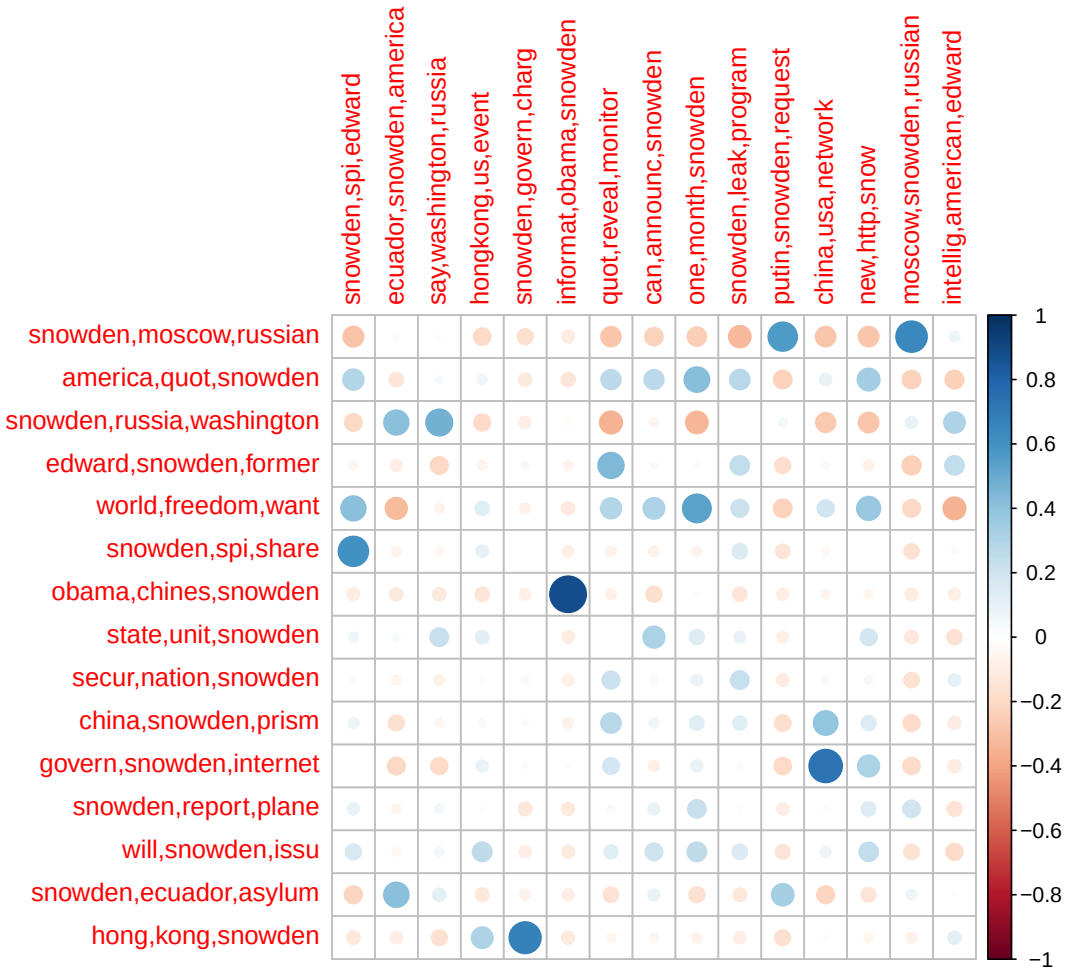

Figure 2: Correlation between topics in both translation approaches. The vertical axis labels topics from full text translation and the horizontal access labels those from document-term matrix translation.

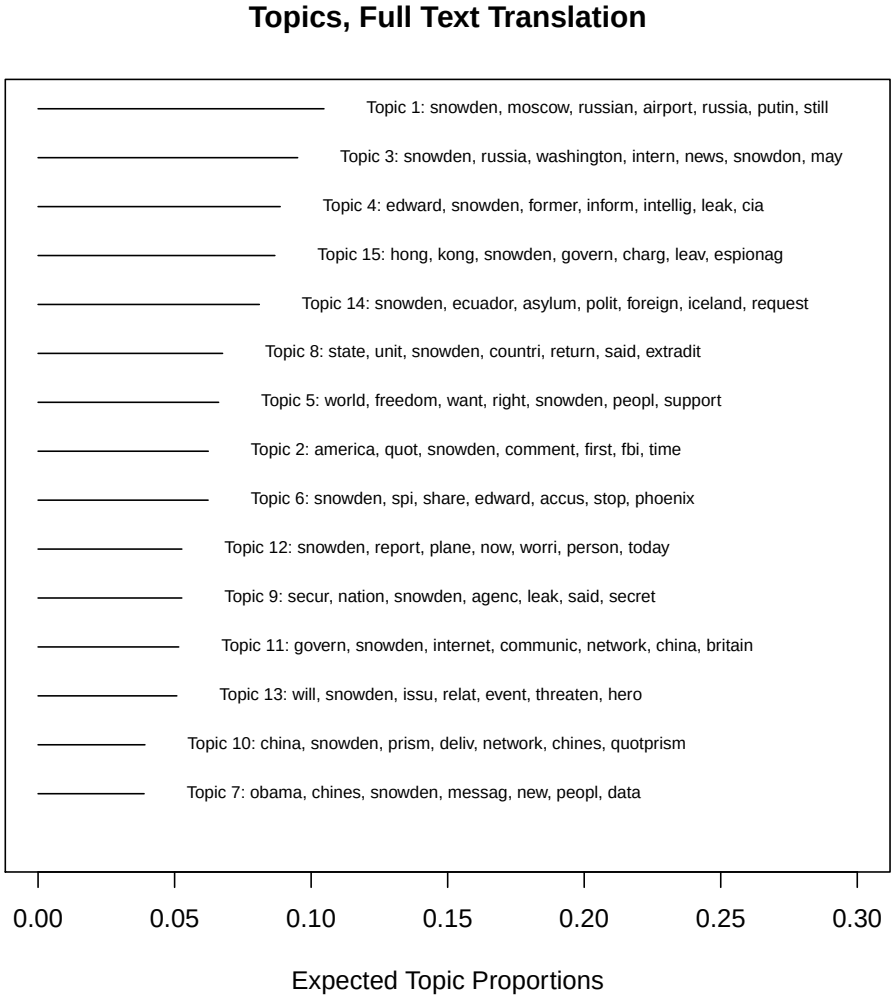

Figure 3: All Topics for Snowden Full Text Translation Model.

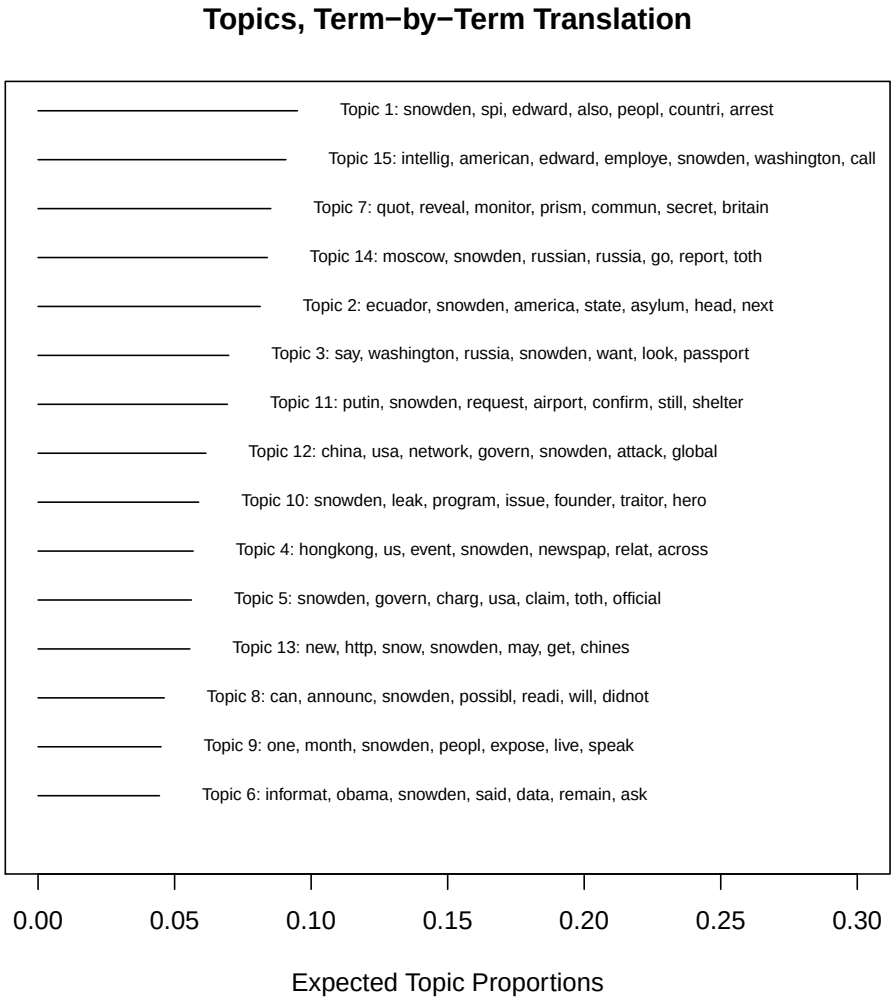

Figure 4: All Topcis for Snowden DTM Translation Model.

Software	txtorg		TM	
	Index	DTM	Corpus Object	DTM
January, 1994 (one month; 6,599 documents)	14.83s	53.23s	34.606s	34.629s
January - December, 1994 (twelve months; 75,303 documents)	174.83s	867.34s	152.566s	1232.507s
January 1994 - December 1998 (five years; 389343 documents)	853.65s	4023.47s	922.255s	FAILED
January 1994 - December 2003 (ten years; 802353 documents)	1643.83s	7235.98s	FAILED	FAILED
Number of languages stemmed	>50		14	
Number of languages stop word removal	>50		15	
Native segmentation support	Chinese, Japanese, Korean		✗	
Index based fast searching	✓		✗	
Outputs associated meta-data	✓		✗	
Multi-corpora management	✓		✗	

Table 3: Results from benchmarking `txtorg` against `TM` on three ranges of New York Times data. All times in seconds. Document length will have an effect on run times. FAILED indicates that R either froze or threw a memory allocation error.

A direct consequence of the index based system is speed in producing the DTM. To demonstrate the speed advantages of an index based system, we benchmark `txtorg` against the commonly recommended `R` package, `TM` (Feinerer et al., 2008).[13] `TM` provides a framework for preprocessing, in which corpora are read into `R`, after which they may be preprocessed and then converted into a DTM object. Though not perfectly analogous, we compare the time necessary to create a corpus object in `TM` to the time necessary to create an index in `txtorg`, and compare the time necessary to create a DTM object in `TM` to that necessary to write a DTM in `txtorg`. We compare the runtime for `TM` to that of `txtorg` on several differently sized corpora from the New York Times. The results from this comparison are shown in Table 3, which clearly demonstrates that while the construction of the index can be nontrivial, it provides a flexible, fast framework for the creation of document-term matrices. Moreover, because users only need to create an index once for any given corpus, moving the computational burden to the indexing stage is prudent, as it makes all subsequent operations faster.[14]

[13] `TM` is a superb package, offering a range of functions useful for text mining. But because it is not an index based system, it suffers in terms of speed and ability to handle large corpora.

[14] `txtorg` outputs several different file formats - primarily, a sparse matrix format and a flat CSV, and hence the corpus can be exported in a form easily read into any major text processing software such as the `Topicmodels` package in `R`.

References

Beauchamp, N. Text-based scaling of legislatures: A comparison of methods with applications to the us senate and uk house of commons.

Blei, D. M. (2012). Probabilistic topic models. *Communications of the ACM 55*(4), 77–84.

Blei, D. M. and J. D. Lafferty (2007). A correlated topic model of science. *The Annals of Applied Statistics 1*(1), 17–35.

Chang, J. (2012). *lda: Collapsed Gibbs sampling methods for topic models.* R package version 1.3.2.

Chang, P.-C., M. Galley, and C. D. Manning (2008). Optimizing chinese word segmentation for machine translation performance. In *Proceedings of the Third Workshop on Statistical Machine Translation*, pp. 224–232. Association for Computational Linguistics.

Converse, P. E., W. E. Miller, J. G. Rusk, and A. C. Wolfe (1969). Continuity and change in american politics: Parties and issues in the 1968 election. *The American Political Science Review 63*(4), 1083–1105.

Elff, M. (2013). A dynamic state-space model of coded political texts. *Political Analysis 21*(2), 217–232.

Feinerer, I., K. Hornik, and D. Meyer (2008, March). Text mining infrastructure in r. *Journal of Statistical Software 25*(5), 1–54.

Gentzkow, M. and J. M. Shapiro (2010). What drives media slant? evidence from us daily newspapers. *Econometrica 78*(1), 35–71.

Grimmer, J. (2010). A bayesian hierarchical topic model for political texts: Measuring expressed agendas in senate press releases. *Political Analysis 18*(1), 1.

Grimmer, J. and B. M. Stewart (2013). Text as data: The promise and pitfalls of automatic content analysis methods for political texts. *Political Analysis 21*(3), 267–297.

Grün, B. and K. Hornik (2011). topicmodels: An R package for fitting topic models. *Journal of Statistical Software 40*(13), 1–30.

Hillard, D., S. Purpura, and J. Wilkerson (2008). Computer-assisted topic classification for mixed-methods social science research. *Journal of Information Technology & Politics 4*(4), 31–46.

Hopkins, D., G. King, M. Knowles, and S. Melendez (2010). Readme: Software for automated content analysis. *Institute for Quantitative Social Science*.

Hopkins, D. J. and G. King (2010). A method of automated nonparametric content analysis for social science. *American Journal of Political Science 54*(1), 229–247.

Hornik, K., J. Rauch, C. Buchta, and I. Feinerer (2012). textcat: N-gram based text categorization. *R package version 0.1-1*.

Jamal, A., R. O. Keohane, D. Romney, and D. Tingley (n.d.). Anti-americanism or anti-interventionism? evidence from the arabic twitter universe. *Perspectives on Politics forthcoming*.

Jensen, J., E. Kaplan, S. Naidu, and L. Wilse-Samson (2012). Political polarization and the dynamics of political language: Evidence from 130 years of partisan speech. *Brookings Papers on Economic Activity 2012*(2), 1–81.

Johnston, A. I. and D. Stockmann (2007). Chinese attitudes toward the united states and americans. *Anti-Americanisms in world politics*, 157–95.

Jurka, T. P., L. Collingwood, A. E. Boydstun, E. Grossman, and W. van Atteveldt (2013). *RTextTools: Automatic Text Classification via Supervised Learning*. R package version 1.4.1.

Kesselman, M. (1966). French local politics: A statistical examination of grass roots consensus. *The American Political Science Review 60*(4), 963–973.

King, G., J. Pan, and M. E. Roberts (2013). How censorship in china allows government criticism but silences collective expression. *American Political Science Review 107*, 1–18.

Klimt, B. and Y. Yang (2004). The enron corpus: A new dataset for email classification research. In *Machine learning: ECML 2004*, pp. 217–226. Springer.

Krippendorff, K. (2012). *Content analysis: An introduction to its methodology*. Sage.

Larkey, L. S., L. Ballesteros, and M. E. Connell (2007). Light stemming for arabic information retrieval. In *Arabic computational morphology*, pp. 221–243. Springer.

Laver, M., K. Benoit, and J. Garry (2003). Extracting policy positions from political texts using words as data. *American Political Science Review 97*(02), 311–331.

Liu, H., F. Han, M. Yuan, J. Lafferty, and L. Wasserman (2012). High-dimensional semiparametric gaussian copula graphical models. *The Annals of Statistics 40*(4), 2293–2326.

Lowe, W. (2008). Understanding wordscores. *Political Analysis 16*(4), 356–371.

Lowe, W. (2013). *Austin: Do things with words*.

Lowe, W. and K. Benoit (2013). Validating estimates of latent traits from textual data using human judgment as a benchmark. *Political Analysis 21*(3), 298–313.

Meinshausen, N. and P. Bühlmann (2006). High-dimensional graphs and variable selection with the lasso. *The Annals of Statistics 34*(3), 1436–1462.

Monroe, B. L., M. P. Colaresi, and K. M. Quinn (2008). Fightin'words: Lexical feature selection and evaluation for identifying the content of political conflict. *Political Analysis 16*(4), 372–403.

Pennebaker, J., M. Francis, and R. Booth (2001). *Linguistic Inquiry and Word Count: LIWC 2001*. Mahway, NJ: Erlbaum Publishers.

Quinn, K., B. Monroe, M. Colaresi, M. Crespin, and D. Radev (2010). How to analyze political attention with minimal assumptions and costs. *American Journal of Political Science 54*(1), 209–228.

Roberts, M. E., B. M. Stewart, and E. Airoldi (n.d.). A topic model for experimentation in the social sciences. *working*.

Roberts, M. E., B. M. Stewart, D. Tingley, and E. M. Airoldi (2013). The structural topic model and applied social science. *Advances in Neural Information Processing Systems Workshop on Topic Models: Computation, Application, and Evaluation*.

Roberts, M. E., B. M. Stewart, D. Tingley, C. Lucas, J. Leder-Luis, S. Gadarian, B. Albertson, and D. Rand (2014). Structural topic models for open-ended survey responses. *American Journal of Political Science 58*(4), 1064 – 1082.

Slapin, J. B. and S.-O. Proksch (2008). A scaling model for estimating time-series party positions from texts. *American Journal of Political Science 52*(3), 705–722.

Spirling, A. (2011). Us treaty making with american indians: Institutional change and relative power, 1784–1911. *American Journal of Political Science*.

Spirling, A. (2012). Comment on 'political polarization and the dynamics of political language: Evidence from 130 years of partisan speech'. *Brookings Papers on Economic Activity 2012*(2), 1–81.

Stockmann, D. (2011). Race to the bottom: media marketization and increasing negativity toward the united states in china. *Political Communication 28*(3), 268–290.

Stone, P., D. Dunphy, M. Smith, and D. Ogilvie (1966). *The General Inquirer: A Computer Approach to Content Analysis*. The MIT Press.

Taddy, M. (2013). Multinomial inverse regression for text analysis. *Journal of the American Statistical Association 108*(503), 755–770.

Tseng, H., P. Chang, G. Andrew, D. Jurafsky, and C. Manning (2005). A conditional random field word segmenter for sighan bakeoff 2005. In *Proceedings of the Fourth SIGHAN Workshop on Chinese Language Processing*, Volume 171. Jeju Island, Korea.

Wallach, H. M. (2006). Topic modeling: beyond bag-of-words. In *Proceedings of the 23rd international conference on Machine learning*, pp. 977–984. ACM.

Zhao, T., H. Liu, K. Roeder, J. Lafferty, and L. Wasserman (2012). The huge package for high-dimensional undirected graph estimation in r. *The Journal of Machine Learning Research 98888*, 1059–1062.