stm: R Package for Structural Topic Models

Margaret E. Roberts UCSD
Brandon M. Stewart Harvard
Dustin Tingley Harvard

Abstract

This vignette demonstrates how to use the Structural Topic Model, **stm**, `R` package. The Structural Topic Model (STM) allows researchers to estimate a topic model which includes document-level meta-data. The **stm** package provides a range of features from model selection to extensive plotting and visualization options.

Keywords: structural topic model, text analysis, LDA, **stm**, R.

1. Introduction

1.1. Method Overview

In this vignette we demonstrate how to use the Structural Topic Model (STM), **stm**, `R` package.[1] Building off of the tradition of generative topic models, such as the Latent Dirichlet Allocation (LDA) (Blei, Ng, and Jordan 2003) and Correlated Topic Model (CTM) (Blei and Lafferty 2007), the Structural Topic Model's key innovation is that it permits users to incorporate *metadata*, defined as information about each document, into the topic model.

The analysis of political data often relies on the quantification of text data. Text data is ubiquitious in social science research: traditional media, social media, survey data, and innumerable other sources contribute to the massive quantity of text in the modern information age. The Structural Topic Model provides users a tool for machine-assisted reading of large text corpora. With the STM, users can model anything from the American National Election Study results (Roberts, Stewart, Tingley, Lucas, Leder-Luis, Gadarian, Albertson, and Rand 2014c) to Chinese newspaper articles (Roberts and Stewart 2014) to online class forums Reich, Tingley, Leder-Luis, Roberts, and Stewart (2014) to Twitter feeds (Lucas, Nielsen, Roberts,

[1]We thank Antonio Coppola, Jetson Leder-Luis, Christopher Lucas, and Alex Storer for various assistance in the construction of this package. Additional details and development version at `structuraltopicmodel.com`

Stewart, Storer, and Tingley 2013).

The goal of the Structural Topic Model is to enable the discovery of topics and the estimation of their relationship to document metadata. Outputs of the model can be used to conduct hypothesis testing about these relationships. This vignette illustrates use of the various components of the package. The design of the package is such that users have a broad array of options to process raw text data, explore and analyze the data and present findings utilizing a range of plotting tools.

The outline of this paper is as follows. In Section 2 we introduce the technical aspects of the STM, including the data generating process and an overview of estimation. In Section 3 we provide examples of how to use the model and the package `stm`, including implementing the model and plots to visualize model output.

2. Model

We begin by providing a technical overview of the STM model. Later in the paper we discuss additional technical details. Like other topic models the STM is a generative model. That means we define a data generating process for each document and then use the data to find the most likely values for the parameters within the model. 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 GLM based on document 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 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.
  $w_{d,n}|z_{d,n}, \beta_{d,k=z} \sim \text{Multinomial}(\beta_{d,k=z})$

Regularizing prior distributions are used for γ, κ and (optionally) Σ which help enhance interpretation and prevent overfitting. To fit the model, we use a semi-collapsed variational Expectation-Maximization algorithm. Figure 1 provides a graphical representation of the model. Further details are provided in additional manuscripts (Roberts, Stewart, Tingley, and Airoldi 2013; Roberts, Stewart, and Airoldi 2014a; Roberts *et al.* 2014c; Lucas *et al.* 2013).[2] In this vignette, we provide only brief interpretations of results, and instead direct readers to the companion papers for complete applications.

3. Using the Structural Topic Model

In this section we demonstrate the basics of using the package.[3] Use of the STM typically proceeds in three key steps:

1. Reading in and processing text data
   (`textProcessor, readCorpus, prepDocuments`)
2. Fitting the Structural Topic Model (`stm, selectModel, manyTopics`)
3. Plotting and inspecting results (`plotModels`,`plot.STM`,`labelTopics`, `estimateEffect`, `plot.estimateEffect`,`findThoughts`, `plotQuote`)

Next we walk through each of these steps to show users how to use the above functions. All of the functions come with help files, and examples, that can be accessed by typing ? and then the function's name.

3.1. Reading in textual data

The first step is to load data into `R`. The **stm** package represents a text corpus in three parts: a

[2] Available here, here, and here.

[3] The stm package depends on a variety of other packages, including **MatrixStats** (Bengtsson 2014), **slam** (Hornik, Meyer, and Buchta 2014), **lda** (Chang 2012), **stringr** (Wickham 2012), **SnowballC** (Bouchet-Valat 2013), **tm** (Meyer, Hornik, and Feinerer 2008), **igraph** (Csardi and Nepusz 2006), **huge** (Zhao, Liu, Roeder, Lafferty, and Wasserman 2013), and **glmnet** (Friedman, Hastie, and Tibshirani 2010).

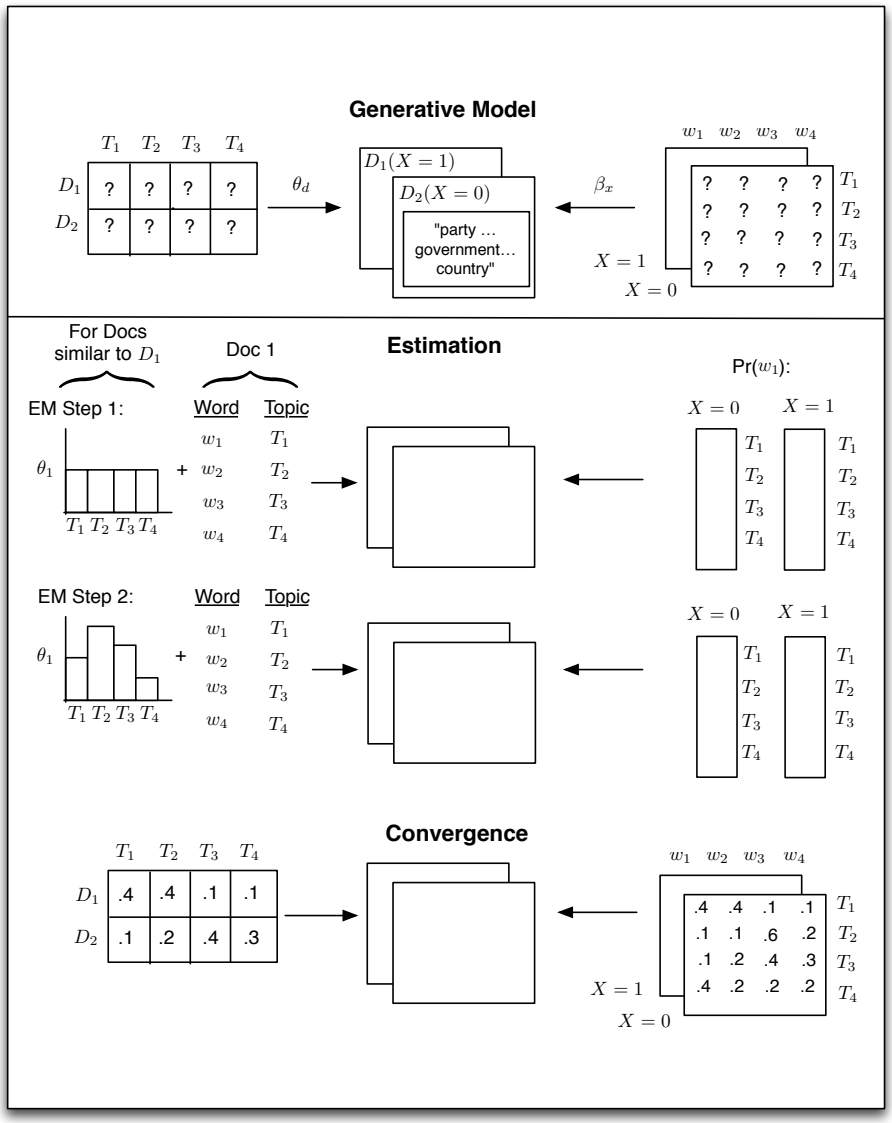

Figure 1: Heuristic description of generative process and estimation of STM.

`documents` list containing word indices and their associated counts,[4] a `vocab` character vector containing the words associated with the word indices and a `metadata` matrix containing document covariates. In this section, we describe utility functions for reading text data into `R` or converting from a variety of common formats. Note that particular care must be taken to ensure that documents and their vocabularies are properly aligned with the metadata.

Reading in data from a "spreadsheet"

For purposes of example within the vignette, we will use a collection of blogposts about American politics that were written in 2008, from the CMU 2008 Political Blog Corpus (Eisenstein and Xing 2010).[5] The blogposts were gathered from six different blogs, American Thinker, Digby, Hot Air, Michelle Malkin, Think Progress, and Talking Points Memo. Each blog has its own particular political bent. The day within 2008 when each blog was written was also recorded. Thus for each blogpost, there is metadata on the day written and the political ideology of the blog in which it was written.

A common way that researchers store textual data alongside covariates related to the text is by having all the data within one spreadsheet, with each row a separate observation and one of the column variables the "textual" data field. The **stm** packages comes with a special function, `textProcessor`, that conveniently reads in data stored in this format and processes the data to ready it for analysis in the **stm** package. For example, users would first read in a csv file using native R functions, or load a pre-prepared dataframe as we do below. Next, they would pass this object through the `textProcessor` function. This function uses a range of features from the `tm` package, such as stemming and stop word removal.

```
> #read in your data that is in a spreadsheet form .csv file here)
> data <- read.csv("poliblogs2008.csv")
> #stemming/stopword removal, etc.
> processed <- textProcessor(data$documents, metadata=data)
> #structure and index for usage in the stm model. Verify no-missingness.
```

[4] A full description of the sparse list format can be found in the help file for `stm`.

[5] The set of blogs is available at http://sailing.cs.cmu.edu/socialmedia/blog2008.html and documentation on the blogs is available at http://www.sailing.cs.cmu.edu/socialmedia/blog2008.pdf. You can find the cleaned version of the data we used for this vignette here: http://goo.gl/tsprNO.

```
> out <- prepDocuments(processed$documents, processed$vocab, processed$meta)
> #output will have object meta, documents, and vocab
> docs <- out$documents
> vocab <- out$vocab
> meta  <-out$meta
```

After reading in the data we suggest using the utility function `prepDocuments()` to process the loaded data to make sure it is in the right format. In particular, the `prepDocuments()` function properly associates metadata with textual data, re-indexes this relationship when textual data fields are blank, or become blank following pre-processing (such as with stop word removal). Please see the help file for this function for more details.

`prepDocuments()` also removes infrequent terms depending on user-set parameter `lower.thresh`. The utility function `plotRemoved()` will plot the number of words and documents removed for different thresholds. For example,the user can use:

```
> plotRemoved(processed$documents, lower.thresh=seq(1,200, by=100))
```

to evaluate how many words and documents would be removed from the dataset at word threshold, which is the minimum number of documents a word needs to appear in in order for the word to be kept within the vocabulary.

Importantly, `prepDocuments()` also will re-index all metadata/document relationships if any changes occur due to processing. After reading in and processing the textual data, it is important to inspect features of the documents and the associated vocabulary list. If users encounter problems with estimation, they should always inspect features such as these, and if using metadata, make sure the metadata has the same number of rows as there are documents. Furthermore, the model does not permit estimation when there are variables used in the model that have missing values.

From here, researchers would be ready to estimate a structural topic model.

Reading in data from other text processing programs

Sometimes researchers will encounter data that is not in a spreadsheet format. The `readCorpus` function includes functionality for loading data from a wide variety of formats, includomg tje

standard R matrix class along with the sparse matrices from the packages **slam** and **Matrix**. Document term matrices created by the popular package **tm** are inherited from the **slam** sparse matrix format and thus are included as a special case.

A program that is helpful for setting up and processing textual data, alongside document metadata, is txtorg (Lucas *et al.* 2013). When using **txtorg**, three separate files are generated. A metadata file, a vocabulary file, and a file with the original documents. The default export format for **txtorg** is the `ldac` sparse matrix format popularized by David Blei's implementation of LDA. The `readCorpus()` function can read in data of this type using the "ldac" option.

3.2. Estimating the STM

Once data is properly imported there will be documents, vocabulary and metadata that can be used for an analysis. In this subsection we illustrate how to estimate the STM.[6]

Ways to use metadata

The key innovation of the STM is a general approach to incorporating metadata into the topic modeling framework. We consider two way that metadata can enter the topic model: topical prevalence and topical content. Metadata covariates for *topical prevalence* allow the observed metadata to affect the frequency with which a topic is discussed. Covariates in *topical content* allow the observed metadata to affect the word rate use within a given topic, that is, how a particular topic is discussed. Estimation for both topical prevalence and content proceeds via the workhorse `stm` function.

Estimation with topical prevalence parameter

In this example, we use the ratings variable (blog ideology) as a covariate in the topic *prevalence* portion of the model with the CMU Poliblog data described above. Each document is modeled as a mixture of multiple topics. Topical prevalence captures how much each topic contributes to a document. Because different documents come from different sources, it is natural then to want to allow this prevalence to vary with metadata that we have about document sources.

[6] We note that all the examples here have a very small number of maximum iterations of the EM algorithm to reduce run time, and so no substantive conclusions should be drawn.

In this example we simply let prevalence be a function of the "ratings" variable, which is coded as either "Liberal" or "Conservative" and the variable "day" which is a integer measure of days running from the first to the last day of 2008. Here a 20 topic STM is estimated. The output from the model, `poliblogPrevFit`, could then be passed through the various functions we discuss below (e.g., `plot.STM`) for inspecting the results.

If a user wishes to specify additional prevalence covariates, she would do so using the standard formula notation common in `R` which we discuss at greater length below. A feature of the `stm()` function is that "prevalence" can be expressed as a formula which can include multiple covariates and factorial or continuous covariates. For example, by using the formula setup we can enter other covariates additively. Additionally users can include more flexible functional forms of continuous covariates, including standard transforms like `log()` etc., as well as `ns()` or `bs()` from the **splines** package. The **stm** package also includes a convenience function `s()` which selects a fairly flexible b-spline basis. In the current example we allow for the variable "date" to be estimated with a spline. As we show later in the vignette, interactions between covariates can also be added using the standard notation for `R` formulas. In summary, for the below example, we enter in the variables additively, but allowing for the day variable, an integer variable measuring which day the blog was posted, to have a non-linear relationship in the topic estimation stage.

```
> poliblogPrevFit <- stm(out$documents,out$vocab,K=20,
+             prevalence =~ rating+ s(day), max.em.its=75,
+             data=out$meta,seed=5926696)
```

The model is set to run for a maximum of 75 EM iterations (`max.em.its()`) using a seed we selected (`seed`). Typically, convergence of the model will be monitored by the change in the approximate bound between EM iterations. Once the bound has a small enough change between iterations, the model is considered converged. To reduce compiling time, in this vignette we do not run the models and instead load a workspace with the models already estimated.

```
>  load(url("http://goo.gl/91KbfS"))
```

3.3. Model Selection and Search

Model selection for a fixed number of number of topics

As with all mixed-membership topic models the posterior is intractable and non-convex, resulting in a multimodal estimation problem which can be sensitive to initialization. Hence users may wish to estimate a many models, each from randomly generated starting values, and then evaluate each model according to some separate standard. The function `selectModel` automates this process to facilitate finding a model with good properties.

Users specify the number of "runs", which in the example below is set to 20. `selectModel` first casts a net where "run" (below 10) models are run for two EM steps, and then models with low likelihoods are discarded. Next the default returns the 20% of models with the highest likelihoods which are then run until convergence or the EM iteration maximum is reached. Notice how that options for the `stm()` function can be passed to `selectModels`, such as `max.em.its`. If users would like to select a larger number of models to be run completely, this can also be set with an option specified in the help file for this function.

```
> poliblogSelect <- selectModel(out$documents,out$vocab,K=20,
+          prevalence =~ rating+s(day), max.em.its=75,
+          data=meta,runs=20,seed=8458159)
```

In order to select a model for further investigation, users must choose from one of the candidate models output from `selectModel()`. To do this `plotModel()` can be used to plot the average *semantic coherence* and *exclusivity* scores for each model (represented by topic numbers) as well as the semantic coherence and exclusivity for each topic.[7] Each of these criteria are calculated for each topic within a model run. The `plotModel()` function calculates the average across all topics for each run of the model and plots these by labeling the model run with a numeral. Often times users will select a model with desirable properties in both dimensions (i.e., models with average scores towards the upper right side of the plot). As shown in Figure 2, the `plotModel()` function also plots each topic's values which helps give a sense of the variation in these parameters.[8]

[7] See Roberts *et al.* (2014a,c) for a discussion of these criteria.

[8] For a given model, the user can plot the semantic coherence and exclusivity scores with the `topicQuality` function.

```
> plotModels(poliblogSelect)
```

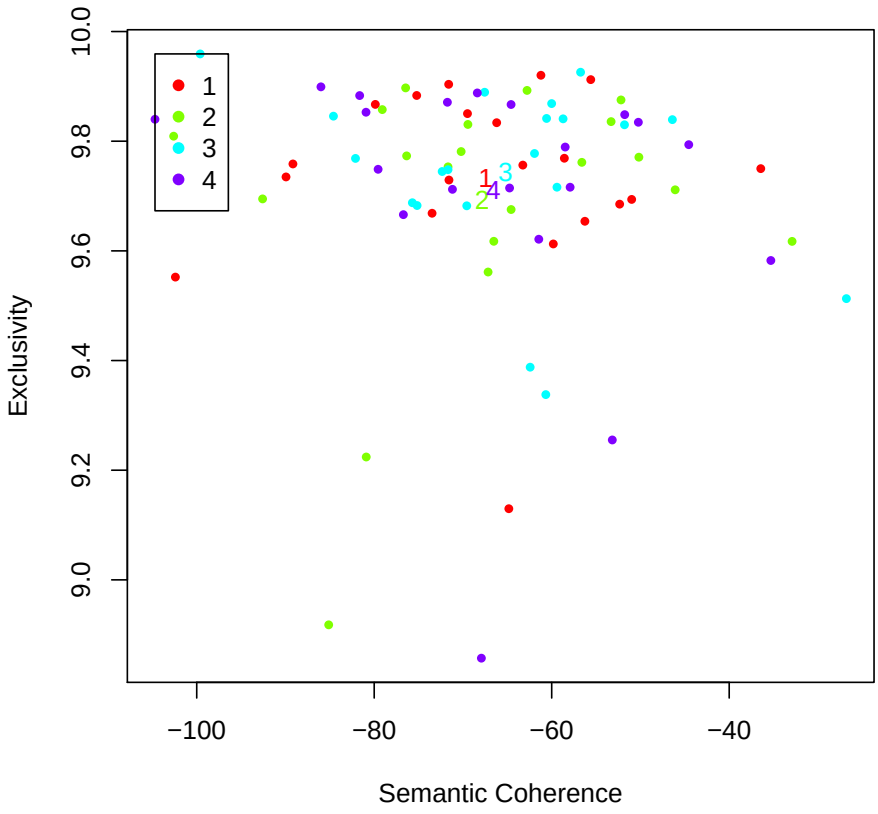

Figure 2: Plot of selectModel results. Numerals represent the average for each model, and dots represent topic specific scores.

Next the user would want to select one of these models to work with. For example, the third model could be extracted from the object that is outputted by `selectModel`.

```
> poliblogPrevFit<-poliblogSelect$runout[[3]] #choose the third model
```

Model search across numbers of topics

STM assumes a fixed user-specified numebr of topics. A data driven approach to selecting the number of topics can be achieved by estimating using `stm` or the `selectModel` tool across multiple different numbers of topics. For example, one could estimate a STM model for 7 and 10 topics. The utility function `manyTopics()` allows the user to specify a range of topic numbers on which she would like to run `selectModel()`. `manyTopics()` will then iterate and run `selectModel()` for each different number of topics. Unlike `selectModel()`, this function automatically selects one run of the model. To do this, it discards runs that are strictly dominated by other model runs on both exclusivity and semantic coherence. Note that the process of extracting results differs from `selectModel()` because results are stored in a list environment.

```
> storage<-manyTopics(out$documents,out$vocab,K=c(7,10),
+         prevalence =~ rating+s(day),data=meta, runs=10)
> #This chooses the output, a single run of STM that was selected,
> #from the runs of the 3 topic model
> t<-storage$out[[1]]
> #This chooses the output, a single run of STM that was selected,
> #from the runs of the 4 topic model
> t<-storage$out[[2]]
```

3.4. Interpreting the STM: Plotting and inspecting results

After choosing a model based on ex-ante criteria, the task of interpretation comes next which is especially crucial for any unsupervised procedure. There are many ways to investigate the output, ranging from inspecting the words associated with topics to the relationship between metadata and topics. To investigate the output of the model the **stm** package provides a number of options.

1. Displaying words associated with topics (`labelTopics`,`plot.STM(,type="labels")`, `sageLabels`,`plot.STM(,type="perspectives")`) or documents highly associated with particular topics (`findThoughts`,`plotQuote`)

2. Plotting relationships between metadata and topics/topical content (`estimateEffect`, `plot.estimateEffect`)

3. Corpus level summaries (`plot.topicCorr`,`plot.STM(,type="summary")`)

Exploring Topics

We next describe two ways that users can explore the topics that have been estimated. The first way is to look at collections of words that are associated with topics. The second way is to read actual documents that are estimated to be highly associated with each topic. Both of these ways should be used.

To explore the words associated with each topic we can use the `labelTopics()` function. For models where a content covariate is included `sageLabels()` can also be used. Both these functions will print to the monitor words associated with each topic. The function by default prints several different types of word profiles, including highest probability words and FREX words.[9] In order to translate these results to a format that can easily be used within a paper, the `plot.stm(,type="labels")` function will print topic words to a graphic device. Notice that in this case, the labels option is specified as the `plot.STM` function has several functionalities that we describe below (the options for "perspectives" and "summary").

```
> labelTopics(poliblogPrevFit, c(1, 7, 10))

Topic 1 Top Words:
          Highest Prob: black, wright, school, ayer, church, white, american
          FREX: wright, church, ayer, black, school, abort, student
          Lift: reverend, jeremiah, church, wright, ayer, cathol, rev
          Score: wright, ayer, church, black, school, abort, jeremiah
```

[9]For more information on FREX and high probability rankings, see Roberts *et al.* (2013, 2014a,c); Lucas *et al.* (2013). For more information on score, see the LDA R package, http://cran.r-project.org/web/packages/lda/lda.pdf. For more information on lift, see Taddy (2013b).

```
Topic 7 Top Words:
          Highest Prob: palin, biden, sarah, romney, joe, governor, huckabe
          FREX: palin, sarah, romney, huckabe, biden, alaska, mitt
          Lift: palin, sarah, huckabe, rudi, romney, giuliani, mitt
          Score: palin, romney, sarah, biden, huckabe, alaska, mitt
Topic 10 Top Words:
          Highest Prob: bush, presid, said, hous, administr, report, white
          FREX: bush, presid, secretari, cheney, administr, rove, meet
          Lift: thinkfast, cheney, dana, rove, bush, dick, inaugur
          Score: bush, presid, administr, cheney, said, hous, rove
```

To read documents that are highly associated with topics the `findThoughts` function can be used. This function will print to the user the documents highly associated with each topic.[10] Reading example documents is helpful in understanding the content of a topic and interpreting its meaning.

In this example, for expositional purposes, we restrict the length of the documents to just plot the first 250 characters. We see that Topic 1 describes the discussion of Jeremiah Wright that occurred around the most recent election. Topic 7 discusses Sarah Palin and other vice presidential candidates, with some discussion of other conservative presidential candidates such as Huckabee or Romney. Topic 10 discusses the Bush administration.

To print example documents to a graphics device `plotQuote` can be used. The results are displayed in Figure 3).

```
> thoughts1<-findThoughts(poliblogPrevFit, texts=shortdoc,
+                         n=2, topics=1)$docs[[1]]
> thoughts7 <- findThoughts(poliblogPrevFit, texts=shortdoc,
+                           n=2, topics=7)$docs[[1]]
> thoughts10 <- findThoughts(poliblogPrevFit, texts=shortdoc,
+                            n=2, topics=10)$docs[[1]]
```

[10] The `theta` parameter in the stm object output has the posterior probability that this function uses.

```
> par(mfrow = c(2, 2),mar=c(.5,.5,1,.5))
> plotQuote(thoughts1, width=40, main="Topic 1")
> plotQuote(thoughts7, width=40, main="Topic 7")
> plotQuote(thoughts10, width=40, main="Topic 10")
```

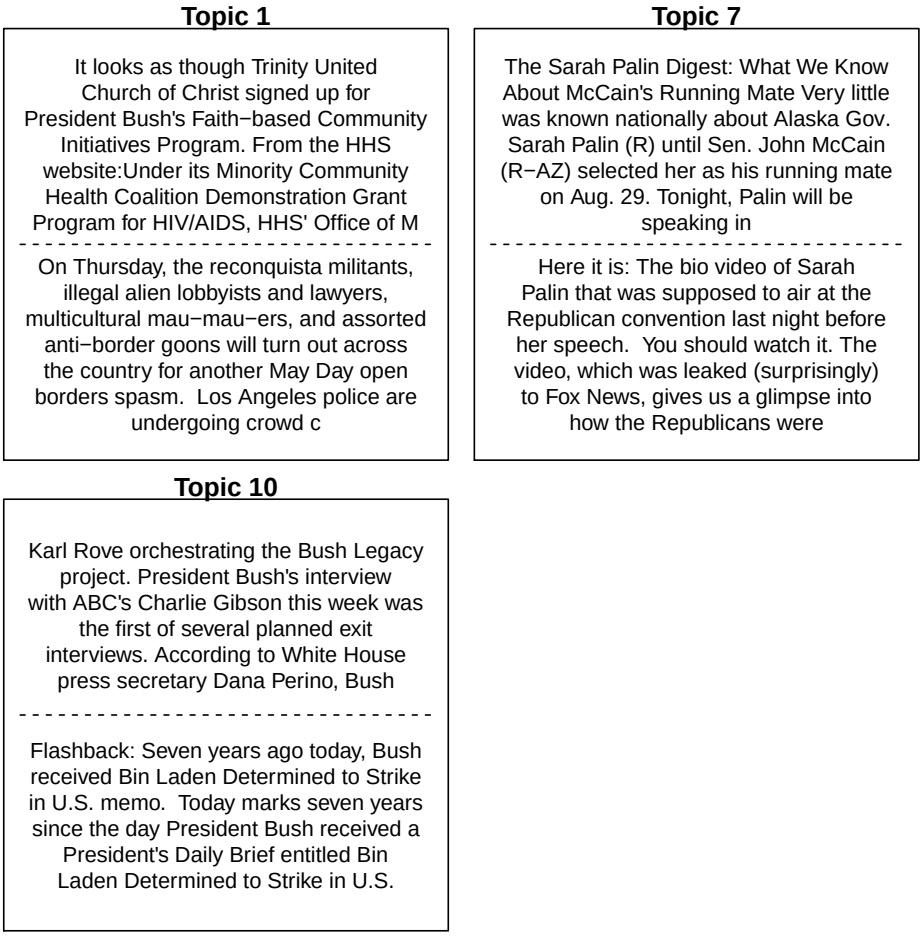

Figure 3: Example documents highly associated with topic 1, 7, and 10.

Plotting Topic/Metadata relationships

We also provide a rich suite of plotting functions that lets users explore the relationship between their metadata and topical prevalence or content.

Estimation Before plotting, we have to run `estimateEffect` in order to simulate a set of parameters which can then be plotted. These parameters include the expected proportion of a document that belongs to a topic as a function of a covariate, or a first difference type estimate, where topic prevalence for a particular topic is contrasted for two groups (e.g., liberal versus conservative). `estimateEffect` uses the method of composition to calculate uncertainty in the parameters fitted by the linear model of the topic proportions on the covariates. In the below example this is run on topics 1 and 2. `estimateEffect` should be run and saved before plotting because it can be time intensive to calculate uncertainty estimates and/or because users might wish to plot different quantities of interest using the same simulated parameters.[11] The output can then be plotted. In this example we use a string variable and it is required that for this function that we first convert it into a factor variable.

```
> meta$rating<-as.factor(meta$rating)
> prep <- estimateEffect(1:20 ~ rating+s(day),poliblogPrevFit,
+          meta=meta, uncertainty = "Global")
```

The syntax of the `estimateEffect` function is designed so users specify the set of topics they wish to use for estimation, and then a formula for metadata of interest. After the necessary method of composition simulations are done particular estimate strategies and standard plot design features can be used by calling the `plot.estimateEffect` function. First, users must specify the variable that they wish to use for calculating an effect. If there were multiple variables specified in `estimateEffect`, then all other variables are held at their sample median. Second users must specify the type of uncertainty calculation. The default is "global", which will incorporate estimation uncertainty of the topic proportions into the uncertainty estimates using the method of composition. If users do not propagate the full amount of uncertainty, e.g., to speed up computational time, they can choose `uncertainty="None"` which will re-

[11] The help file for this function describes several different ways for uncertainty estimate calculation, some of which are much faster than others.

sult in narrower confidence intervals because it will not include the additional estimation uncertainty.

Plotting Users must also select the type of quantity that is to be plotted. When the covariate of interest is binary, or users are interested in a particular contrast, the method="difference" option will plot the change in topic proportion shifting from one specific value to another. Figure 4 gives an example. For factor variables, users may wish to plot the marginal topic proportion for each of the levels ("pointestimate").

We see Topic 1 is strongly used by conservatives compared to liberals, while Topic 7 is close to the middle but still conservative-leaning. Topic 10, the discussion of Bush, was largely associated with liberal writers, which is in line with the observed trend of conservatives distancing from Bush after his presidency.

Notice how the function makes use of standard labeling options available in the native plot() function. This allows the user to customize labels and other features of their plots. We note that in the package we leverage generics for the plot functions. As such, one can simply use `plot` instead of writing out the full extension (e.g., in Figure 4 one could use `plot()` instead of `plot.estimateEffect`). For expositional purposes in this vignette, we include the entire extension.

When users have variables that they want to treat continuously, users can choose between assuming a linear fit or using splines. In the previous example, we allowed for the day variable to have a non-linear relationship in the topic estimation stage. We can then plot its effect on topics. In Figure 5, we plot the relationship between time and the vice presidential topic, topic 7. The topic peaks when Sarah Palin became John McCain's running mate at the end of August in 2008.

Topical Content

We can also plot the influence of covariates included in topical content. A topical content variable allows for the vocabulary used to talk about a particular topic to vary. First, the STM must be fit with a variable specified in the content option. In the below example, ratings serves this purpose. It is important to note that this is a completely new model, and so the actual topics may differ in both content and numbering compared to the previous example

```
> plot.estimateEffect(prep, covariate = "rating", topics = c(1, 7, 10),
+         model=poliblogPrevFit, method="difference",
+         cov.value1="Liberal",cov.value2="Conservative",
+         xlab="More Conservative ... More Liberal",
+         main="Effect of Liberal vs. Conservative",
+         xlim=c(-.1,.1), labeltype = "custom",
+         custom.labels = c('Jeremiah Wright', 'Sarah Palin',
+                           'Bush Presidency'))
```

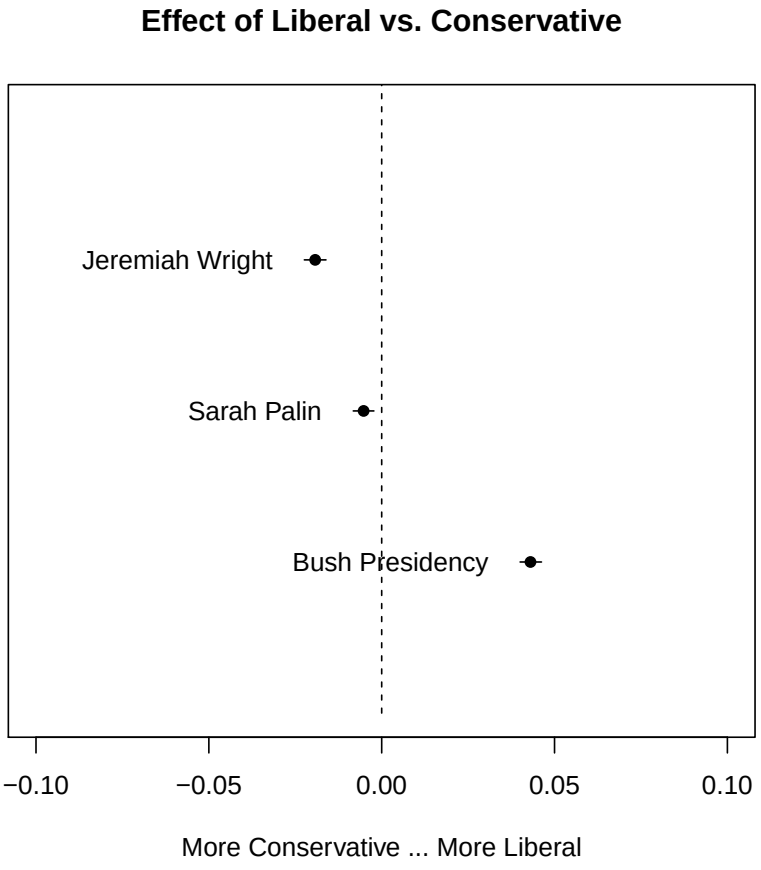

Figure 4: Graphical Display of Topical Prevalence Contrast.

```
> plot.estimateEffect(prep, "day", method="continuous", topics=7, model=z,
+ printlegend=FALSE, xaxt="n", xlab="Time (2008)")
> monthseq <- seq(from=as.Date("2008-01-01"),
+                 to=as.Date("2008-12-01"), by="month")
> monthnames <- months(monthseq)
> axis(1,
+      at=as.numeric(monthseq)-min(as.numeric(monthseq)),
+      labels=monthnames)
```

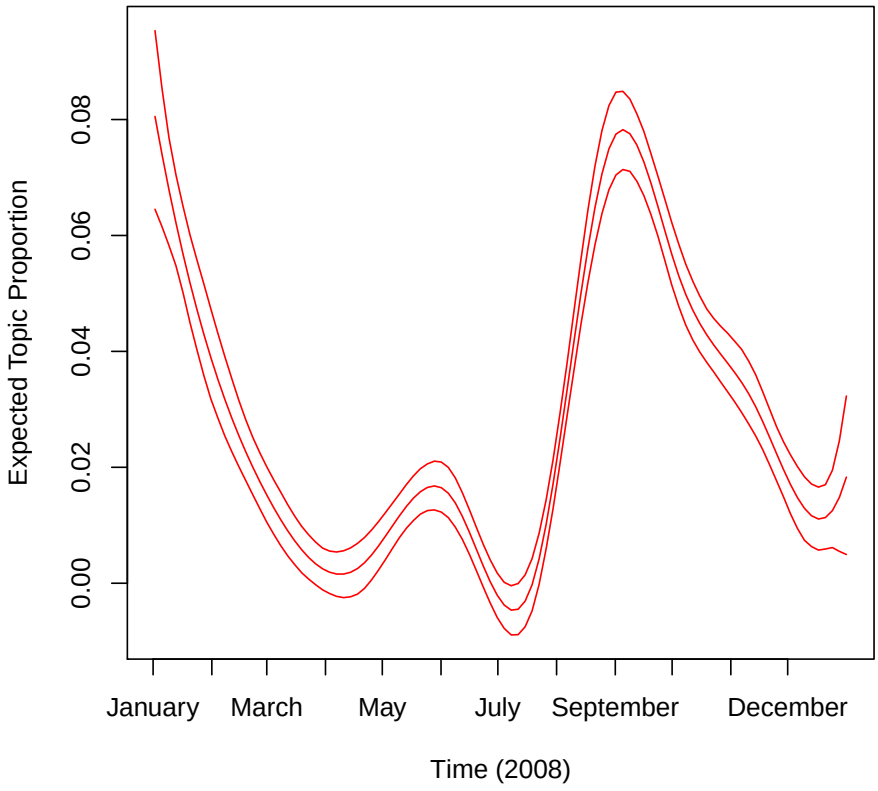

Figure 5: Graphical Display of Topic Prevalence. Topic 7 prevalence is plotted as a smooth function of day, holding rating at sample median.

where no content covariate was used.

```
> poliblogContent <- stm(out$documents,out$vocab,K=20,
+          prevalence =~ rating+ s(day), content=~rating,
+          max.em.its=75, data=out$meta,seed=5593453)
```

Next, the results can be plotted using the `plot.STM(,type="perspectives")` function. This functions shows which words within a topic are more associated with one covariate value versus another. In Figure 6, vocabulary differences by ratings is plotted for topic 10. Topic 10 is related to Guantanamo. Its top words were "guantanamo, techniqu, interrog, cia, detaine, tortur, detent." However, Figure 6 lets us see how liberals and conservatives talk about this topic differently. In particular, liberals emphasized "torture" whereas conservatives emphasized how the detainees were "terrorists."[12]

This function can also be used to plot the contrast in words across two topics.[13] To show this we go back to our original model that did not include a content covariate and we contrast topic 9 (Iraq war) and 10 (Bush presidency). We plot the results in Figure 7.

The cloud function can also be used to visualize topics. For example, Figure 8 displays a word cloud of the most probable words in a topic related to the vice presidential candidates in the 2008 election.

Covariate Interactions

Another modification that is possible in this framework is to allow for interactions between covariates such that we allow for one variable to "moderate" the effect of another variable. In this example, we re-estimated the STM to allow for an interaction between day (entered linearly) and ratings. Then in `estimateEffect()` we include the same interaction. This allows us in `plot.estimateEffect` to have this interaction plotted. We display the results in Figure 9 for topic 1 (Jeremiah Wright). We observe that liberals never really talked about

[12] At this point you can only have a single variable as a content covariate, although that variable can have any number of groups. It cannot be continuous. Note that the computational cost of this type of model rises quickly with the number of groups and so it may be advisable to keep it small.

[13] This plot calculates the difference in probability of a word for the two topics, normalized by the maximum difference in probability of any word between the two topics.

```
> plot.STM(poliblogContent,type="perspectives", topics=10)
```

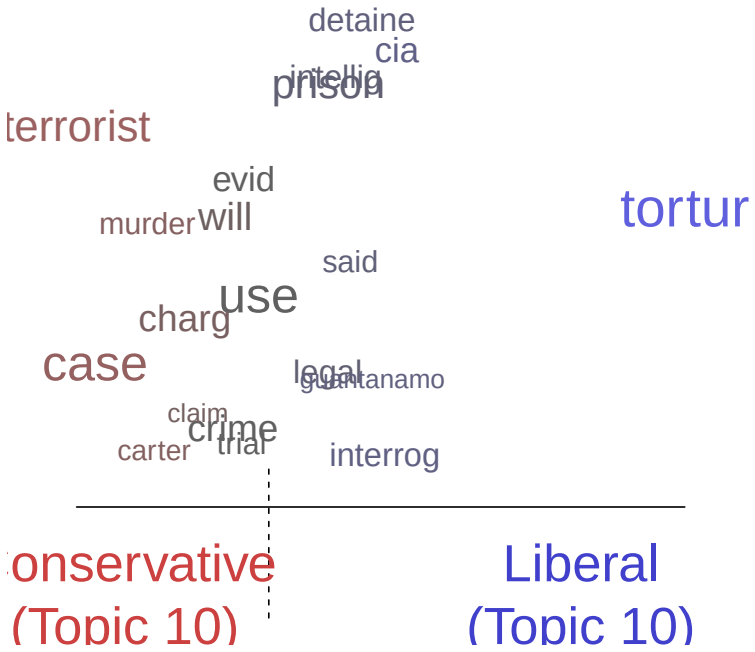

Figure 6: Graphical Display of Topical Perspectives.

```
> plot.STM(poliblogPrevFit,type="perspectives", topics=c(9, 10))
```

Figure 7: Graphical Display of Topical Contrast between Topics 9 and 10.

```
> cloud(poliblogPrevFit, topic=7)
```

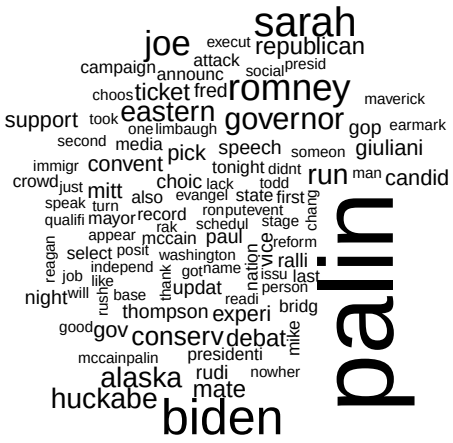

Figure 8: WordCloud Display of Topics.

this topic, whereas for conservatives this was initially high and then declined.[14]

```
> poliblogInteraction <- stm(out$documents,out$vocab,K=20,
+         prevalence =~ rating* day, max.em.its=75,
+         data=out$meta,seed=5926696)
```

More details are available in the help file for this function.[15].

Corpus level plotting

Corpus level visualization can be done in several different ways. The first relates to the expected proportion of the corpus that belongs to each topic. This can be be plotted using `plot.stm(,type="summary")`. An example from the political blogs data is given in Figure 10. We see, for example, that the Sarah Palin/Vice President topic (7) is actually a relatively minor proportion of the discourse. The most common topic, topic 3, is a general topic full of words that bloggers commonly use, and therefore is not very interpretable. The words printed are the top three words associated with the topic.

Users can also plot features of the corpus as a whole. First, the Structural Topic Model permits correlations between topics. Positive correlations between topics indicate that both topics are likely to be discussed within a document. These can be visualized using `plot.topicCorr()`. The user can specify a correlation threshold. If two topics are correlated above that threshold, than those two topics are considered linked. After calculating the links between topics, `plot.topicCorr` produces a layout of topic correlations using a force-directed layout algorithm. `plot.topicCorr` has several options that are described in the help file. We can use the correlation graph to observe the connection between such as topics 9 (Iraq War) and 10 (Bush).

```
> mod.out.corr<-topicCorr(poliblogPrevFit)
```

[14]Note that the ability to plot interactions is somewhat limited and only supports interactions with a binary effect modification covariate and does not support interactions with a spline.

[15]An additional option is the use of local local regression (loess). In this case, because multiple covariates are not possible a separate function is required, `plotTopicLoess`, which contains a help file for interested users.

```
> prep <- estimateEffect(c(1) ~ rating*day, poliblogInteraction,
+          metadata=meta, uncertainty="None")
> plot.estimateEffect(prep, covariate="day", model=poliblogInteraction,
+          method="continuous",xlab="Days", moderator="rating",
+          moderator.value="Liberal", linecol="blue", ylim=c(0,.08),
+          printlegend=F)
> plot.estimateEffect(prep, covariate="day", model=poliblogInteraction,
+          method="continuous",xlab="Days", moderator="rating",
+          moderator.value="Conservative", linecol="red", add=T,
+          printlegend=F)
> legend(0,.08, c("Liberal", "Conservative"),
+        lwd=2, col=c("blue", "red"))
```

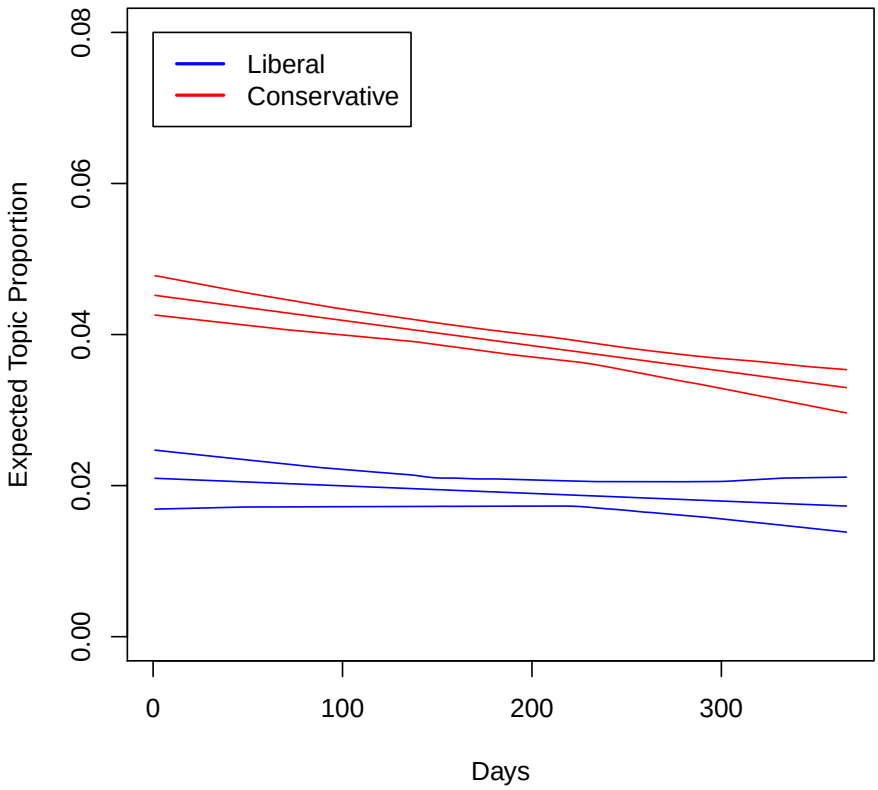

Figure 9: Graphical Display of Topical Content allowing for interaction between day of blog post and liberal versus conservative interaction. Topic 1 prevalence is plotted as linear function of day, holding the rating at either 0 (Liberal) or 1 (Conservative). Were other variables included in the model, they would be held at their sample medians.

```
> plot.STM(poliblogPrevFit,type="summary", xlim=c(0,.4))
```

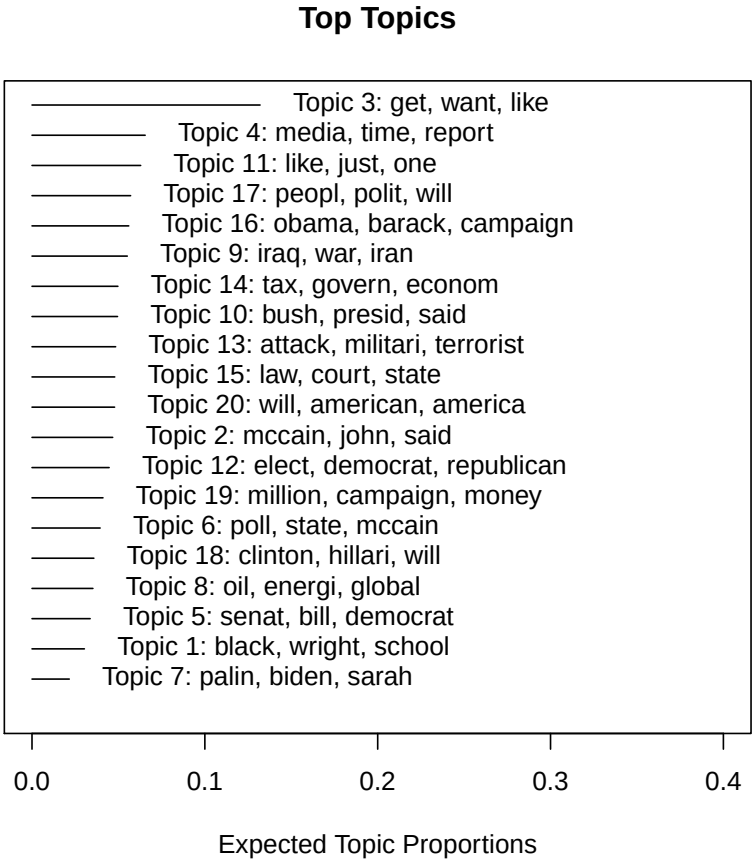

Figure 10: Graphical Display of Estimated Topic Proportions.

```
> plot.topicCorr(mod.out.corr)
```

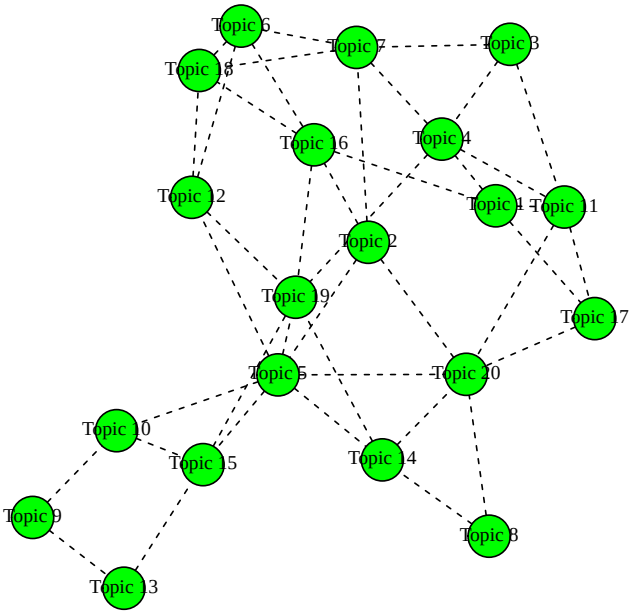

Figure 11: Graphical Display of Topic Correlations.

3.5. Diagnostic Tests

Topic estimation is fundamentally imprecise, as estimation in topic model space requires both an a priori number of topics input by the user, and furthermore an optimization in a space with multiple solutions. Due to the intractability underlying the computation of topic models, we rely on external analytics of our model to understand its unique tradeoffs between competing parameters. The STM package contains a variety of tools that can be used to evaluate the quality of the model as well as the user's choice of number of topics and of metadata selected for inclusion.

1. Post-estimation Permutation Checks

Any statistical procedure can be abused and STM is no different. One concern that we have heard is that users will simply search out covariate topic relationships that are the strongest. A related concern is the concern that by combining the measurement model with the estimation of an effect, the user is 'baking in' the conclusion. In the appendix of Roberts *et al.* (2014c) we address this concern using both a simulation and also a permutation test approach. We have built in a function for conducting permuation tests using binary prevalence covariates.[16] The `permutationTest` function takes a formula containing a single binary covariate (and optionally other controls) and runs a permutation test where, rather than using the true assignment, the covariate is randomly assigned to a document with probability equal to its empirical probability in the data. After each shuffle of the covariate the same STM model is estimated at different starting values using the same initialization procedure as the original model, and the effect of the covariate across topics is calculated. Next the function records two quantities of interest across this set of "runs" of the model. The first records the absolute maximum effect of the permuted covariate across all topics. The second records the effect of the (permuted) covariate on the topic in each additional stm run which is estimated to be the topic closest to the topic of interest. The object returned from `permutationTest` can then be passed to `plot.STMpermute` for plotting.

2. Checks for Multi-modality

Another diagnostic that should be completed while running the STM is checking to see how

[16] Future work could extend this to other types of covariates.

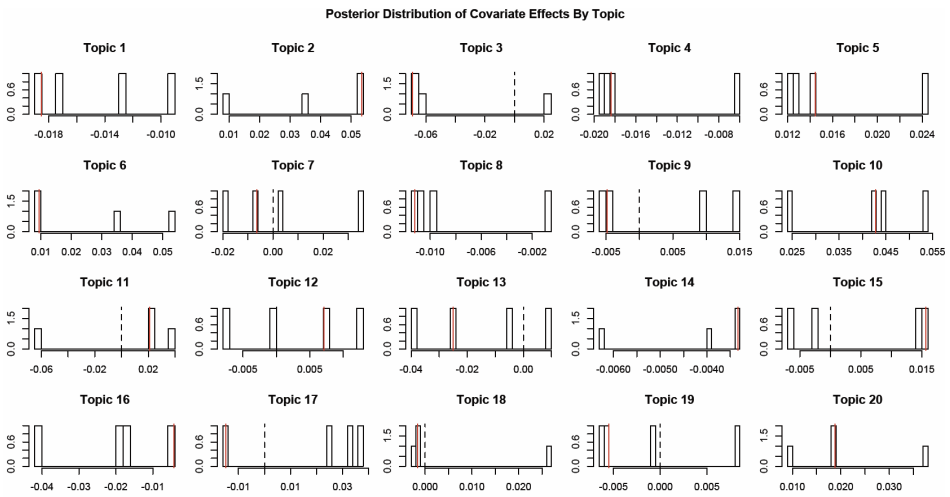

Figure 12: Plot of Liberal-Conservative differences effect across four runs.

multi-modal the model of interest is. We provide a suite of methods to assess multi-modality, and we refer the reader to Roberts, Stewart, and Tingley (2014b) for an explanation of all of them. For purposes of example, we check how robust the covariate effects we explore earlier are to multimodality within the four original models that we ran. The function `multiSTM` aligns topics across models. The function `plot.MultimodDiagnostic` plots the effects across topics and models. As you can see in Figure 12, we find that only some of topics in our first STM model are robust across models. If the topic of interest is not robust across models, we suggest looking more closely at the topic's most probable words to try to understand where these differences orginate.

3. Held-out likelihood estimation

Sometimes users will want to compare model specifications to see how well the model does predicting words within the document. The `stm` package contains two different functions to aid with held-out likelihood estimation. Held-out likelihood estimation is the estimation of the probability of words appearing within a document when those words have been removed from the document in the estimation step (Blei *et al.* 2003). Similar to cross-validation, when some of the data is removed from estimation and then later used for validation, the held-out likelihood helps the user assess the model's prediction performance.

We provide two different functions for the user to complete heldout likelihood estimation. The

first, `make.heldout` produces a document set where some of the words within the documents have been removed. The user can then run a STM model on the documents that have the missing words. The second, `eval.heldout`, evaluates the heldout likeihood for missing words based on the model run on the heldout documents.

4. Residuals Checks

Users can also test the assumptions of the model within the package through the function `residuals`. This function implements residual checks described in Section 4.2 of Taddy (2012), testing for overdispersion of the variance of the multinomial within the data generating process of STM. As described in Taddy (2012), if the residuals are overdispered, it could be that more topics are needed to soak up some of the extra variance. While no fool-proof method has been developed to choose the number of topics, both the residual checks and held-out likelihood estimation can be useful for indications of the number of topics to choose.

In addition to these functions one can also explore if there words that are extremely highly associated with a single topic via the `checkBeta` function.

4. Changing Basic Estimation Defaults

In this section we overview how to change basic defaults in estimation. We start by discussing how to chose among different methods for intializing model parametes. We then discuss how to set and evaluate convergence criteria. Next we overview a method for accelerating convergence in document sets with many documents. Finally we discuss some variations on content covariate models which allow the user to control model complexity.

4.1. Initialization

As with most topic models, the objective function maximized by STM is multimodal. This means that the way we choose the starting values for the variational EM algorithm can affect our final solution. We provide three methods of initialization accessed using the argument `init.type`, Latent Dirichlet Allocation via collapsed Gibbs sampling (`init.type="LDA"`), a Spectral algorithm for Latent Dirichlet Allocation (`init.type="Spectral"`), and random starting values (`init.type="Random"`).

LDA is the default option and uses a few passes of collapsed Gibbs sampling to initialize the algorithm. The exact parameters for this initialization can be set using the argument `control`.The Spectral option initializes using a moment based estimator for LDA due to Arora, Ge, Halpern, Mimno, Moitra, Sontag, Wu, and Zhu (2012). In contrast to the LDA and random initializations this approach is deterministic. It performs extremely well particularly for larger document sets. Because the spectral algorithm needs to form a square matrix with dimensions of the lenght of the vocabulary, it is best used in settings where the vocabulary is under 5000 terms. Finally the random algorithm draws the initial state from a Dirichlet distribution. This is included primarily for completeness and in general th other two strategies should be preferred. Roberts *et al.* (2014b) provides details on these initialization methods and provides a study of their performance. In general Spectral outperforms LDA which in turn outperforms Random initialization.

Each time the model is run the random seed is saved in the output object under `settings$seed`. This can be passed to the `seed` argument of `stm` to replicate the same starting values.

4.2. Convergence Criteria

Estimation in the STM proceeds by variational EM. Convergence is controlled by relative change in the variational objective. Denoting by ℓ_t the approximate variational object at time t, convergence is declared when the quantity $\ell_t - \ell_{t-1}/\text{abs}(\ell_{t-1})$ drops below tolerance. The default tolerance is 1e-5 and can be changed using the `emtol` argument.

The argument `max.em.its` sets the maximum number of iterations. If this threshold is reached before convergence is assessed a message will be printed to the screen. The default of 500 iterations is simply a general guideline. A model which fails to converged can be restarted using the `model` argument in `stm`. See the documentation for `stm` for more information.

The default is to have the status of iterations print to the screen. The `verbose` option turns printing to the screen on and off.

During the E-step the algorithm prints one dot for every 1% of the corpus it completes and announces completion along with timing information. Printing for the M-Step depends on the algorithm being used. For models without content covariates M-step estimation should be nearly instantaneous. For models with content covariates and the algorithm prints dots to

indicate progress. The exact interpretation of the dots differs with the choice of model (see the help file for more details).

By default every 5th iteration will print a report of top topic and covariate words. The `reportevery` option sets how often these reports are printed.

Once a model has been fit, convergence can easily be assessed by plotting the variational bound as in Figure 13.

4.3. Accelarating Convergence

When the number of documents is large convergence in topic models can be slow. This is because each iteration requires a complete pass over all the documents before updating the global parameters. To accelerate convergence we can split the documents into several equal sized blocks and update the global parameters after each block. The option `ngroups` specifies the number of blocks, and setting it equal to an integer greater than one turns on this functionality.

Note that setting the `ngroups` value to a large number can dramatically increase the memory requirements of the function. Thus as the number of blocks is increased we are trading off memory for computational efficiency.

4.4. SAGE

The Sparse Additive Generative (SAGE) model conceptualizes topics as sparse deviations from a corpus-wide baseline (Eisenstein, Ahmed, and Xing 2011). While computationally more expensive this can sometimes produce higher quality topics . Whereas LDA will tend to assign rare words exclusively to one topic, the regularization of the SAGE model ensures that words only load onto topics when they have sufficient counts to overwhelm the prior. In general this means that SAGE topics will tend to have fewer words that distinguish them from other topics, but those words are more likely to be meaningful. Importantly for our purposes the SAGE framework makes it straightforward to add covariate effects into the content portion of the model.

Covariate-Free SAGE While SAGE topics are enabled automatically when using a covariate in the content model they can also be used even without covariates. To activate SAGE

```
> plot(poliblogPrevFit$convergence$bound,type="l",
+      ylab="Approximate Objective",
+      main="Convergence")
```

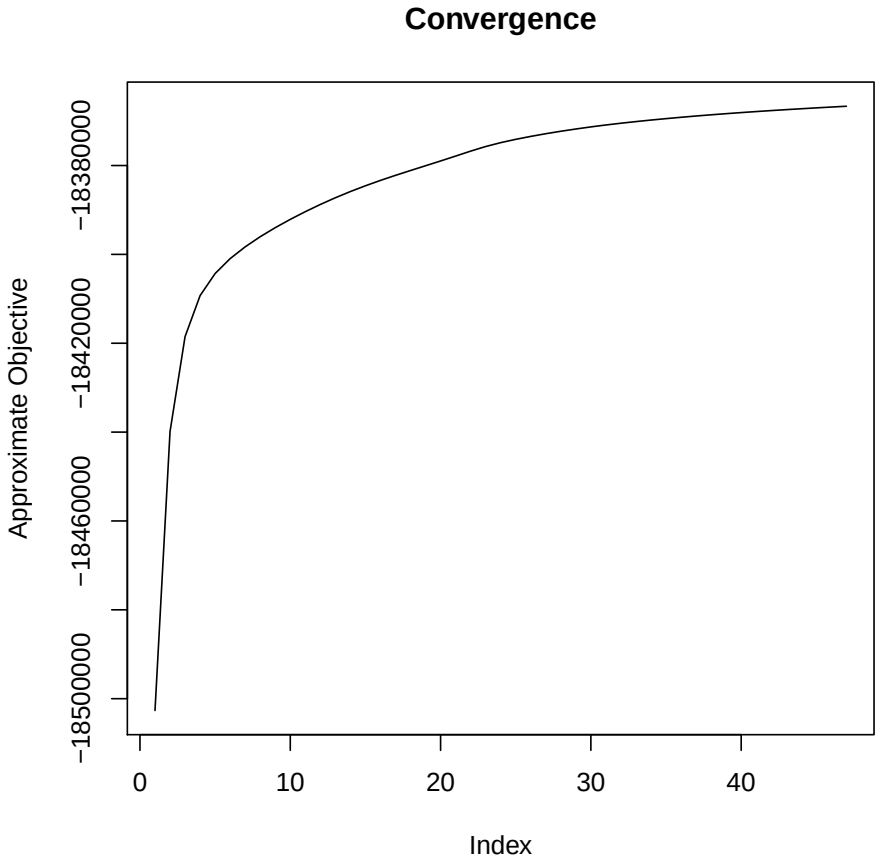

Figure 13: Graphical Display of Convergence.

topics simply set the option `LDAbeta=FALSE`.

Covariate-Topic Interactions By default when a content covariate is included in the model, we also include covariate-topic interactions. In our political blog corpus for example this means that the probability of observing a word from a Conservative blog in Topic 1 is formed by combining the baseline probability, the Topic 1 component, the Conservative component and the Topic 1 - Conservative interaction component.

Users can turn off interactions by specifying the option `interactions=FALSE`. This can be helpful in settings where there isn't sufficient data to make reasonably inferences about all the interaction parameters. It also reduces the computational intensity of the model.

5. Alternate Priors

In this section we overview options for altering the prior structure in the `stm` function. We highlight the alternatives and provide intuition for the properties of each option. We have chosen the default settings to represent the choices that we expect will perform the best in the majority of cases and thus changing these settings should only be necessary if the defaults are not performing well.

5.1. Changing Estimation of Prevalence Covariate coefficients

The user can choose between two options: "Pooled" and "L1". The difference between these two is that the "L1" option can induce sparsity in the coefficients (i.e. many are set exactly to zero) while the "Pooled" esitmator is computationally more efficient. In practice we recommend the default "Pooled" estimator unless the prevalence covariates are very high dimensional (such as a factor with hundreds of categories).

"Pooled" is the default option and estimates a model where the coefficients on topic prevalence have a zero-mean Normal prior with variance given a broad inverse-gamma hyperprior. This provides moderate shrinkage towards zero but does not induce sparsity.

You can also choose `gamma.prior="L1"` which uses the `glmnet` package (Friedman *et al.* 2010) to allow for grouped penalties between the L1 and L2 norm. In these settings we estimate a regularization path and then select the optimal shrinkage parameter using a user-tunable

information criterion. By default selecting the L1 option will apply the L1 penalty selecting the optimal shrinkage parameter using AIC. The defaults have been specifically tuned for the STM but almost all the relevant arguments can be changed through the `control` argument. Changing the `gamma.enet` parameter by specifying `control=list(gamma.enet=.5)` allows the user to choose a mix between the L1 and L2 norms. When set to 1 (as by default) this is the lasso penalty, when set to 0 its the ridge penalty. Any value in between is a mixture called the elastic net.

5.2. Changing Covariance Matrix Prior

The `sigma.prior` argument is a value between 0 and 1 defaulting to 0. The update for the covariance matrix is formed by taking the convex combination of the diagonalized covariance and the MLE with weight given by the prior. Thus by default we are simply maximizing the likelihood. When `sigma.prior=1` this amounts to setting a diagonal covariance matrix. This argument can be useful in settings where topics are at risk of becoming too highly correlated. However, in extensive testing we have come across very few cases where this was needed.

5.3. Changing the Content Covariate Prior

The `kappa.prior` option provides two sparsity promoting priors for the content covariates. The default is `kappa.prior="L1"` and uses `glmnet` and the distributed multinomial formulation of Taddy (2013a). The core idea is to decouple the update into a sequence of independent L1-regularized poisson models with plugin estimators for the document level shared effects. See Roberts *et al.* (2014a) for more details on the estimation procedure. The regularization parameter is set automatically as detailed in the `stm` help file.

To maintain backwards compatability we also provide estimation using a scale mixture of Normals where the precisions τ are given improper Jeffreys priors $1/\tau$. This option can be accessed by setting `kappa.prior="Jeffreys"`. We caution that this can be much slower than the default option.

There are over a dozen additional options documented in `stm` for altering additional components of the prior, most of them focusing on the content covariate model.

6. Conclusion

The **stm** package provides a flexible integration of document metadata and topic modeling. This vignette provides an overview of use and features. We encourage users to consult the extensive help files for more details, as well as read the companion papers that illustrate the application of this method.

The development of the structural topic model is ongoing, and future research promises a variety of tools for better models and more user-friendly results. Software for the visualization of the STM is ongoing, with development of a web browser-based visualization tool underway. Such interactive visualizations will allow for further exploration of topics and the interactions between topics and covariates.

Furthermore, in performing these intensive machine learning computations, there are always gains in efficiency to be had, both in theoretical optimality and in applied programming practice. The STM is undergoing constant streamlining and revision towards faster, more optimal computation. As corpus sizes increase, the STM will also increase in capacity to handle more documents and more varied metadata.

References

Arora S, Ge R, Halpern Y, Mimno D, Moitra A, Sontag D, Wu Y, Zhu M (2012). "A practical algorithm for topic modeling with provable guarantees." *arXiv preprint arXiv:1212.4777*.

Bengtsson H (2014). *matrixStats: Methods that apply to rows and columns of a matrix*. R package version 0.8.14, URL http://CRAN.R-project.org/package=matrixStats.

Blei DM, Lafferty JD (2007). "A correlated topic model of science." *The Annals of Applied Statistics*, pp. 17–35.

Blei DM, Ng A, Jordan M (2003). "Latent Dirichlet allocation." *Journal of Machine Learning Research*, **3**, 993–1022.

Bouchet-Valat M (2013). *SnowballC: Snowball stemmers based on the C libstemmer UTF-8 library*. R package version 0.5, URL http://CRAN.R-project.org/package=SnowballC.

Chang J (2012). *lda: Collapsed Gibbs sampling methods for topic models*. R package version 1.3.2, URL http://CRAN.R-project.org/package=lda.

Csardi G, Nepusz T (2006). "The igraph software package for complex network research." *InterJournal*, **Complex Systems**, 1695. URL http://igraph.org.

Eisenstein J, Ahmed A, Xing E (2011). "Sparse additive generative models of text." In *Proceedings of ICML*, pp. 1041–1048.

Eisenstein J, Xing E (2010). "The CMU 2008 Political Blog Corpus."

Friedman J, Hastie T, Tibshirani R (2010). "Regularization paths for generalized linear models via coordinate descent." *Journal of statistical software*, **33**(1), 1.

Hornik K, Meyer D, Buchta C (2014). *slam: Sparse Lightweight Arrays and Matrices*. R package version 0.1-31, URL http://CRAN.R-project.org/package=slam.

Lucas C, Nielsen R, Roberts M, Stewart B, Storer A, Tingley D (2013). "Computer assisted text analysis for comparative politics." *Working Paper*.

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

Reich J, Tingley D, Leder-Luis J, Roberts ME, Stewart BM (2014). "Computer Assisted Reading and Discovery for Student Generated Text." *Working Paper*.

Roberts ME, Stewart BM (2014). "What the Big Picture Reveals About Governments: Scalable Text Analysis of Millions of Newspapers in China." *Working paper*.

Roberts ME, Stewart BM, Airoldi EM (2014a). "Structural Topic Models."

Roberts ME, Stewart BM, Tingley D (2014b). "Navigating the Local Modes of Big Data: The Case of Topic Models." *Technical report*, Working Paper. Export BibTex Tagged XML.

Roberts ME, Stewart BM, Tingley D, Airoldi EM (2013). "The Structural Topic Model and Applied Social Science."

Roberts ME, Stewart BM, Tingley D, Lucas C, Leder-Luis J, Gadarian S, Albertson B, Rand D (2014c). "Structural topic models for open-ended survey responses." *American Journal of Political Science*.

Taddy M (2013a). "Distributed Multinomial Regression." *arXiv preprint arXiv:1311.6139*.

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

Taddy MA (2012). "On Estimation and Selection for Topic Models." *Journal of Machine Learning Research*.

Wickham H (2012). *stringr: Make it easier to work with strings*. R package version 0.6.2, URL http://CRAN.R-project.org/package=stringr.

Zhao T, Liu H, Roeder K, Lafferty J, Wasserman L (2013). *huge: High-dimensional Undirected Graph Estimation*. R package version 1.2.5, URL http://CRAN.R-project.org/package=huge.

Affiliation:

Margaret E. Roberts

Department of Political Science

University of California, San Diego

Social Sciences Building 301

9500 Gillman Drive, 0521, La Jolla, CA, 92093-0521

E-mail: meroberts@ucsd.edu

URL: http://www.margaretroberts.net

Brandon M. Stewart

Department of Government

Harvard University

1737 Cambridge St, Cambridge, MA, USA

E-mail: bstewart@fas.harvard.edu

URL: http://scholar.harvard.edu/bstewart

Dustin Tingley

Department of Government

Harvard University

1737 Cambridge St, Cambridge, MA, USA

E-mail: dtingley@gov.harvard.edu

URL: http://scholar.harvard.edu/dtingley