stm: R Package for Structural Topic Models

Margaret E. Roberts Harvard

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 using information, metadata, about the documents. The **stm** 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**, `R` package.[1] Building off of the tradition of generative topic models, such as the Latent Dirichlet Allocation (Blei, Ng, and Jordan 2003) and Correlated Topic Model (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 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. Previous uses of the model span from the analysis of newspaper articles and their relationship to time and open ended responses in survey data. 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 analyze the data and present findings utilizing a range of plotting tools.[2]

[1] We thank Jetson Leder-Luis, Christopher Lucas, and Alex Storer for various assistance in the construction of this package.

[2] Some of the functionalities are illustrated as examples sourced via `demo(STMbasics)` and `demo(STMoptions)` and all functions come with rich help files.

2. Estimation

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 Σ which help enhance interpretation and prevent overfitting. To fit the model, the Expectation and Maximization algorithm is used, given that we are estimating a set of latent variables. Further details are provided elsewhere (Roberts, Stewart, Tingley, and Airoldi 2013b; Roberts, Stewart, and Airoldi 2013a; Roberts, Stewart, Tingley, Lucas, Leder-Luis, Gadarian, Albertson, and Rand 2014; Lucas, Nielsen, Roberts, Stewart, Storer, and Tingley 2013).[3] In this vignette, we do not engage in interpretation of specific substantive results, and instead direct readers to the companion papers for discussions.

[3] Available here, here, and here.

3. Using the Structural Topic Model

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

1. Tools for 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`. Unlike standard quantitative text analysis techniques, textual analysis requires more preparations prior to analysis. In this section, we describe several different methods for loading data into `R` depending on the format of the textual data. These functions are provided in order to make the structural topic model as accessible as possible. A key aspect of this step for the `STM`, as compared to textual analysis using other techniques, is the need to make sure documents and their vocabularies are properly associated with their metadata.

Reading in data from a "spreadsheet"

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.

```
> library(stm)
> temp<-textProcessor(documents=gadarian$open.ended.response,metadata=gadarian)

Building corpus...
Converting to Lower Case...
Removing stopwords...
Removing numbers...
Removing punctuation...
Stemming...
Creating Output...

> meta<-temp$meta
> vocab<-temp$vocab
> docs<-temp$documents
> out <- prepDocuments(docs, vocab, meta)
> docs<-out$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. It also removes infrequent terms, or too frequent terms, depending on user set parameters. Importantly,prepDocuments() also will re-index all metadata/document relationships if any changes occur due to processing.

Reading in data from other text processing programs

The readCorpus function is helpful for reading in data produced in other text processing/analysis programs. The structure of these data sets are not the simple "spreadsheet" format like in the previous example. For example, 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 function `readCorpus()` function can be called to help import this type of data, in order to import the vocabulary file from `textorg` and set it up for analysis by the `stm` model. Please see the help file for this function for more details and additional options such as reading in data in Blei's LDA-C format.

Another way to load in data is to use data structured for use in other packages, like the popular LDA package.

Here we load the political Blog Corpus from the LDA Package.

```
> library(lda)
> data(poliblog.documents) # these are the word counts
> data(poliblog.vocab)     # =words in the vocabulary
> data(poliblog.ratings)   # rating of blog as
>                          # conservative or liberal
```

This is a subset of the CMU Poliblog corpus by Ahmed and Xing. cite

The LDA package uses a slightly different format for the vocabulary. Specifically it indexes words from 0 using the convention in C. We can convert this over to a more R-like 1-index using `prepDocuments()`, as well as perform other recoding options of metadata. Once again, the next step is to reassign the documents object from the output of `prepDocuments()` to its own object.

```
> poliblog.ratings <- as.factor(ifelse(poliblog.ratings==-100,
+                                      "Liberal",
+                                      "Conservative"))
> out <-prepDocuments(poliblog.documents, poliblog.vocab, poliblog.ratings)
> poliblog.documents <- out$documents
> poliblog.ratings <- out$meta
> poliblog.vocab <- out$vocab
```

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.

```
> length(poliblog.documents) #there are 773 documents

[1] 773

> length(poliblog.vocab)     #and 1290 items in the vocab

[1] 1290
```

In the poliblogs data there is only one covariate. However, the STM is a multivariate method. To illustrate this we use a corpus from Matt Taddy's **textir** package which is drawn from Gentzkow and Shapiro (2010). This corpus contains 1000 phrases from the 2005 Congressional record. Each "document" is a different legislator's transcript for that year. We use this data set as an example because gives us access to lots of interesting covariates which enables us to showcase the flexibility of the **stm** package. To read in the data we use the `readCorpus()` function specifying the type as `Matrix` because the data is stored as a sparse matrix from the `Matrix` package. This gives us our document-term matrix, vocabulary, and metadata.

```
> library(textir)
> data(congress109)
> temp <- readCorpus(congress109Counts, type="Matrix")
> documents.gs <- temp$documents
> vocab.gs <- temp$vocab
> metadata.gs <- congress109Ideology #this is our metadata.
> out <-prepDocuments(documents.gs, vocab.gs, metadata.gs)

Removing Words Due to Frequency
Your corpus now has 529 documents and 992 words.
```

```
> documents.gs <- temp$documents
> vocab.gs <- temp$vocab
> metadata.gs <- out$meta
> rm(temp)
```

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.[4]

Ways to use metadata

Typically, metadata about documents has not been incorporated into the topic model. The key innovation of the STM is to incorporate metadata in the the analysis. In the STM there are two ways metadata about each document can be included in the 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. We first show how to leverage metadata for understanding topical content.

Estimation with topical prevalence parameter

In this example we use the ratings variable as a covariate in the topic prevalence portion of the model. Each document can be 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.

In this example we simply let prevalence be a function of the poliblog.ratings variable. Here topic prevalence is a linear function of poliblog.ratings and a five topic STM is estimated. The output from the model, mod.out, could then be passed through the various functions we discuss below for inspecting the results. If a user wishes to specify additional prevalence

[4] 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.

covariates, they would do so using the standard formula notation common in R which we discuss at greater length below.

```
> poliblogPrevFit <- stm(poliblog.documents,poliblog.vocab,K=10,
+                        prevalence=~poliblog.ratings)
```

For example purposes only, the model is set to only run for five EM iterations. 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.

3.3. Model Selection

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 10. `selectModel` first 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 model 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, such as `max.em.its`.

```
> poliblogSelect <- selectModel(poliblog.documents,poliblog.vocab,K=10,
+                               prevalence=~poliblog.ratings, runs=20)
```

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 topic (represented by topic numbers) as well as the semantic coherence and exclusivity for each topic.[5] 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

[5] See Roberts et al. (2013a, 2014) for a discussion of these criteria.

```
> plotModels(poliblogSelect)
```

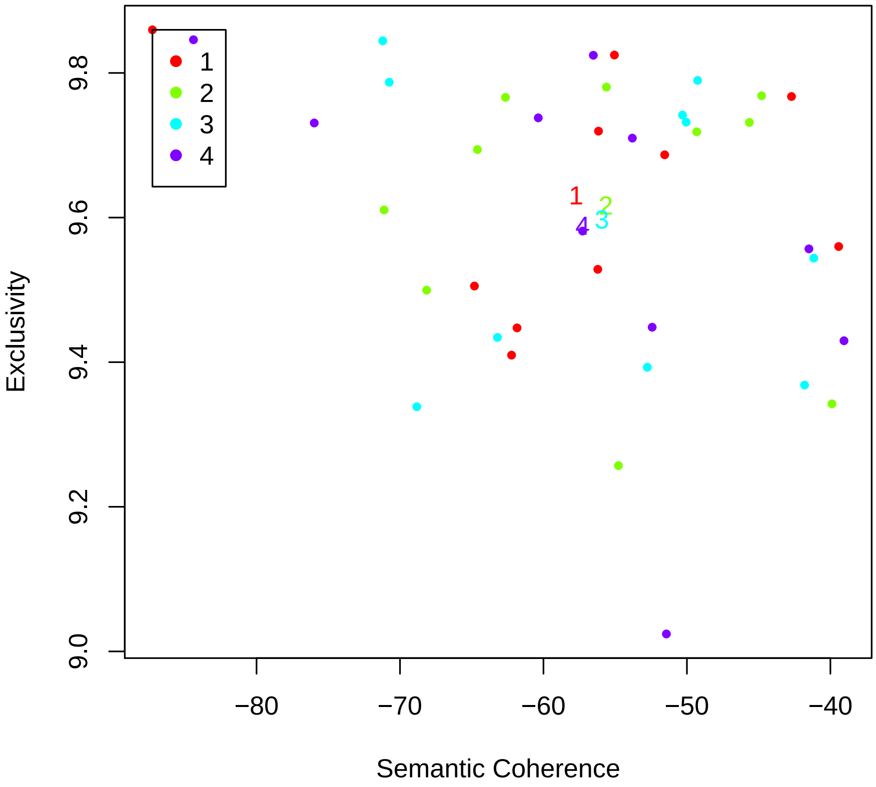

Figure 1: Plot of selectModel results.

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 1, the `plotModel()` function also plots each topic's values which helps give a sense of the variation in these parameters.[6]

3.4. Interpreting the STM: Plotting and inspecting results

After estimating a 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

[6] The utility function `manyTopics()` allows the user to specify a range of topic numbers that they would like to run `selectModel` on. Please consult the help file for information on this function.

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")`) 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. In 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. This 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.[7] 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.

```
> labelTopics(poliblogPrevFit,topics=c(1,3))

Topic 1 Top Words:
          Highest Prob: people, like, just, dont, get, think, know
          FREX: dont, left, get, think, youre, really, tell
          Lift: straight, youre, sit, happens, happening, hot, hell
```

[7] For more information on FREX and high probability rankings, see Roberts et al. (2013b,a, 2014); 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 (2012).

```
          Score: youre, hes, going, think, know, guy, dont
Topic 3 Top Words:
          Highest Prob: voters, vote, percent, north, indiana, poll, carolina
          FREX: indiana, poll, voters, polls, results, percent, north
          Lift: precincts, rasmussen, turnout, poll, indiana, exit, results
          Score: voters, indiana, percent, polls, poll, carolina, rasmussen
```

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. To print example documents to a graphics device `plotQuote` can be used. The help files for both of these functions provide examples.

Plotting Topic/Metadata relationships

The `STM` package comes with a rich suite of plotting functions that lets users explore the relationship between their metadata and topical prevalence or content.

Before plotting, we have to run `estimateEffect` in order to simulate a set of parameters which can then be plotted. `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. The output can then be plotted.

```
> prep <- estimateEffect(1:2 ~ poliblog.ratings,poliblogPrevFit)
```

The syntax of the `estimateEffect` function is designed so users specify the set of topics they wish for estimation to be done on, 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. In this example we only have one variable, ”poliblog.ratings”. Users must

```
> plot.estimateEffect(prep, "poliblog.ratings", model="poliblogFit", method="difference",
+              cov.value1="Liberal",cov.value2="Conservative",
+             xlab="Liberal-Conservative",
+             main="Effect of Liberal vs. Conservative"
+             )
```

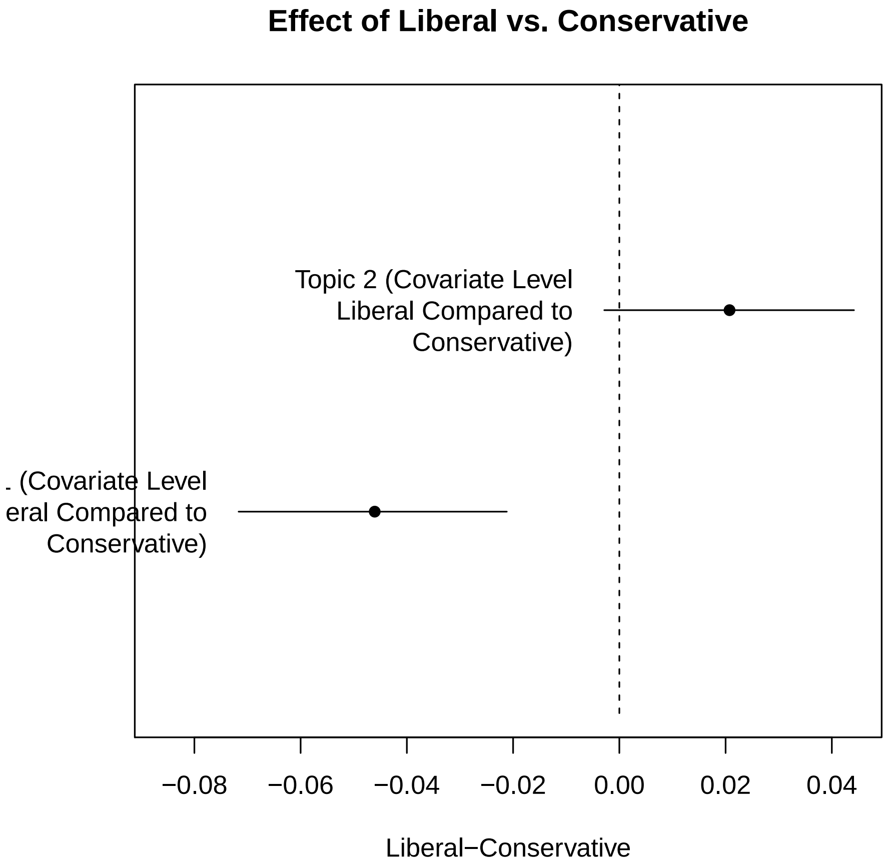

Figure 2: Graphical Display of Topical Prevalence Contrast.

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 value to the other. Figure 2 gives an example. For factor variables, users may which to plot the marginal topic proportion for each of the levels (”pointestimate”).

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 2 one could use `plot()` instead of `plot.estimateEffect`). For expositional purposes in this vignette, we include the entire extension.

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, poliblog.ratings serves this purpose. 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 3, vocabulary differences by poliblog.ratings is plotted for topic 5.[8][9]

In the previous example we had a single binary variable. 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. We showcase this by using the data from Gentzkow and Shapiro (2010) that we previously loaded from the **textir** package. 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(x) which selects a fairly flexible b-spline basis. Interactions between covariates can also be added using the standard notation for `R` formulas. In the below example, we enter in the variables additively, but allowing for the repshare variable to have a non-linear relationship in the topic estimation stage.

```
> congress109Fit <- stm(documents.gs,vocab.gs, K = 25,
+                        prevalence = ~party + chamber + s(repshare),
```

[8] As described in the help file for plot.stm, the perspectives option also can be used to contrast two separate topics even when no content covariate is specified.

[9] 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.

```
> plot.STM(poliblogFit,type="perspectives", topics=5)
```

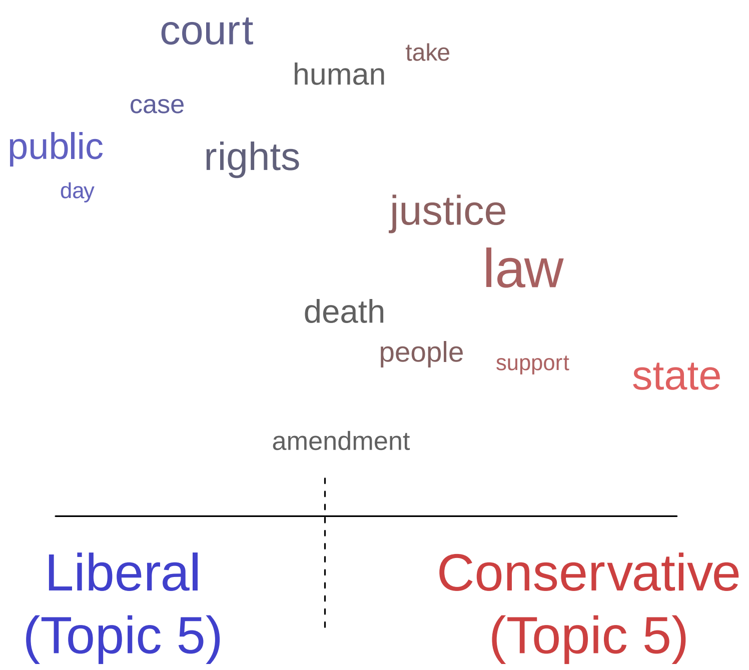

Figure 3: Graphical Display of Topical Perspectives.

```
+                              data = metadata.gs)

> prep <- estimateEffect(c(7) ~ party + chamber + s(repshare),
+            congress109Fit, metadata=metadata.gs)
```

After estimating the STM, researchers can investigate the relationship between topics and a particular covariate. 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 repshare variable to have a non-linear relationship in the topic estimation stage, and restricted estimation to a single topic (topic 7). We can then plot its effect on topics in Figure 4

```
> plot.estimateEffect(prep, covariate="repshare", model=congress109Fit,
+                     method="continuous",printlegend=FALSE)
```

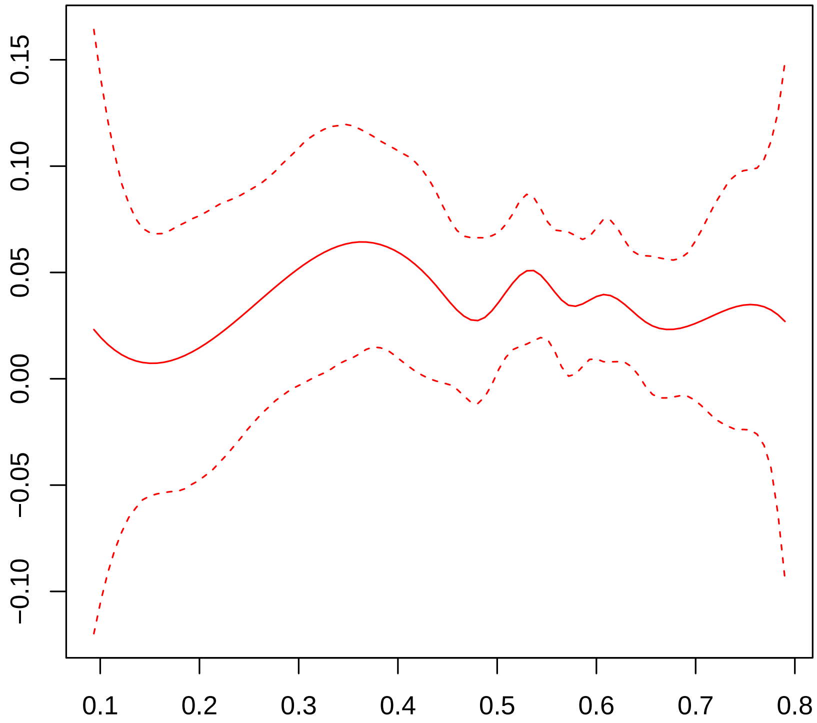

Figure 4: Graphical Display of Topical Content. Topic 7 prevalence is plotted as a smooth function of repshare, holding party and chamber at sample medians.

Another modification that is possible in this framework is to allow for interactions between covariates. In this example, we re-estimated the STM to allow for an interaction between chamber (House or Senate) and repshare. Then in `estimateEffect()` we include the same interaction. This allows us in `plot.estimateEffect` to indicate that our variable of interest is still repshare, but by choose the method "continuous" and specifying values of treat.value and control.value to be the levels we set the moderating variable to, we can plot the effect of reshare by chamber, holding the other covariates at their sample medians.[10]

[10] Note that the ability to plot interactions is somewhat limited and only supports interactions with a binary

```
> prep <- estimateEffect(c(7) ~ party + chamber*repshare,
+          congress109Int, metadata=metadata.gs)
> plot.estimateEffect(prep, covariate="repshare", model=congress109Int,
+                     method="continuous")
```

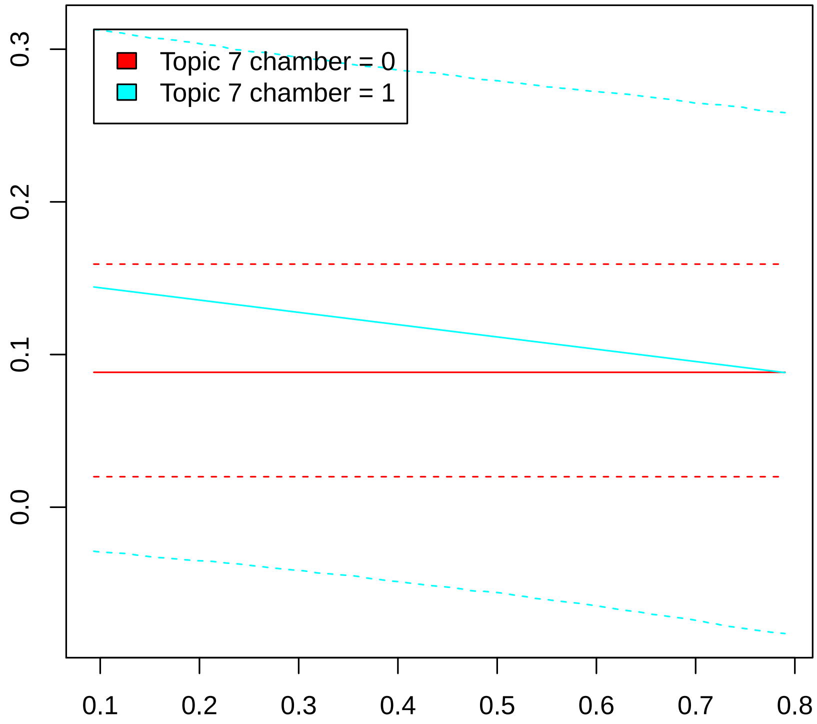

Figure 5: Graphical Display of Topical Content allowing for interaction between chamber and repshare. Topic 7 prevalence is plotted as linear function of repshare, holding chamber at either 0 (H) or 1 (S) and party at sample median.

```
> congress109Int <- stm(documents.gs,vocab.gs, K=10,
+                       prevalence= ~party + chamber*repshare,
+                       data=metadata.gs)
```

covariate for plotting.

```
> plot.STM(congress109Fit,type="summary")
```

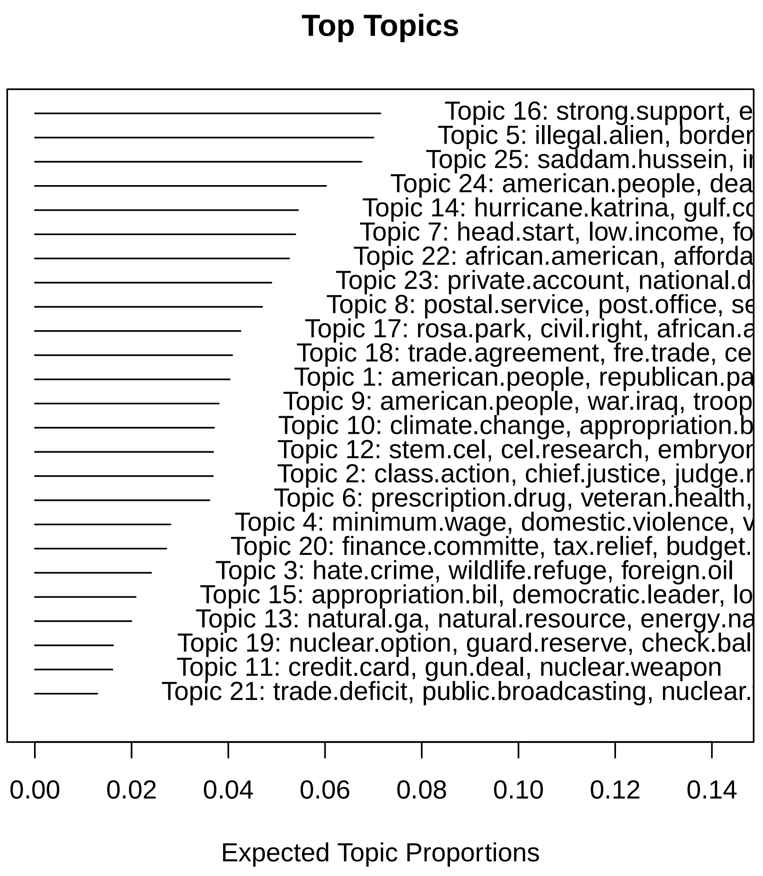

Figure 6: Graphical Display of Estimated Topic Proportions.

More details are available in the help file for this function.[11].

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 Congressional transcript data is given in Figure 6.

Users can also plot features of the corpus as a whole. First, the Structural Topic Model permits

[11] 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.

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

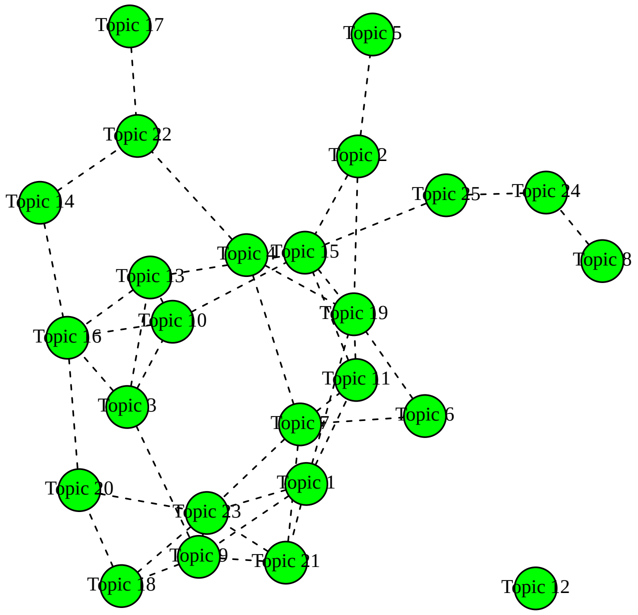

Figure 7: Graphical Display of Topic Correlations.

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.

```
> mod.out.gs.corr<-topicCorr(congress109Fit)
```

4. Changing Basic Estimation Defaults

This section briefly mentions a number of ways to change default estimation settings. These features are useful for more advanced users.

4.1. EM iterations

The EM basis of the STM means that an iterative approach is taken, with convergence evaluated according to a criterion. Users can limit the total number of iterations, or change the convergence threshold. The convergence threshold declares convergence when the relative change in the approximate bound drops below 1e-5 by default.

`max.em.its` controls the maximum number of EM iterations and `em.converge.thresh` controls the convergence threshold.

```
> mod.out <- stm(documents, vocab, K=10,
+                prevalence= ~party, data=metadata,
+                max.em.its=2, em.converge.thresh=1e-2)
```

4.2. Initialization

As with any EM based estimation strategy, starting values are required. In order to change how starting values are chosen, init.type has several different options including `LDA`, `DMR` (Dirichlet Multinomial Regression Topic Model), `Random` and `User`). `LDA` is the default, although we provide alternatives for initialization and users should estimate several models to investigate their semantic properties.

```
> mod.out <- stm(documents, vocab, K=10,
+                prevalence= ~party, data=metadata,
+                init.type="Random", max.em.its=5)
```

4.3. Printing

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

```
> mod.out <- stm(documents, vocab, K=10,
+                prevalence= ~party, data=metadata,
+                max.em.its=2, verbose=FALSE)
> # Understanding the Default Printing
> #   Estep: it prints one dot for every floor(N/100)
> #          documents as it goes through the estep
> #          so its roughly one dot \approx 1% complete.
> #          Text announces completion.
> #   Mstep: it prints one dot for each parameter vector
> #          update. So if you have K=10 topics, A=2
> #          aspects with interactions.  Then you have
> #          32 dots per mstep pass.
> #         Each pass gets its own line.
> #   Every 5th iteration it prints a report of top topic
> #          and aspect words.
> #          (this can be changed via the reportevery
> #          argument)
> #   Each iteration it prints the approximate bound
```

4.4. SAGE

The basic idea of the Sparse Additive Generative (SAGE) model is to conceptualize 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.[12]

Some defaults are hidden within the ... but can be changed by simply passing the appropriate

[12] This can be done even without a content covariate.

argument. Users can turn on SAGE topics.

```
> mod.out <- stm(documents, vocab, K=10,
+                 prevalence= ~party, data=metadata,
+                 max.em.its=2, LDAbeta=FALSE)
```

Setting LDAbeta=FALSE turns on Sparse additive topics even without covariates.

4.5. content-covariate/topic interactions

Users may wish to turn off interactions between the content-covariate / topic interactions for theoretical reasons. This would be the case when the analyst believes that a covariate may have an effect on word-useage but only one that is constant across all topics. It can also be useful in cases where inclusion of the interaction causes the different versions of the topic to diverge too substantially to still be theoretically the same concept.

To do this set interactions to FALSE per below.

```
> mod.out <- stm(documents, vocab, K=10,
+                 prevalence= ~party, data=metadata,
+                 content=party,
+                 max.em.its=2, interactions=FALSE)
```

4.6. Topic Printing

A default is to print the estimated topics as as the EM estimation progresses. The `reportevery` options changes the frequency that this printing happens.

```
> mod.out <- stm(documents, vocab, K=10,
+                 prevalence= ~party, data=metadata,
+                 max.em.its=2, reportevery=1)
```

4.7. Keeping the History

Setting the option `keephistory` to `TRUE` causes the parameters estimated at each EM step to retained.

```
> mod.out <- stm(documents, vocab, K=10,
+                prevalence= ~party, data=metadata,
+                max.em.its=2, keephistory=TRUE)
```

This can be useful for diagnostic purposes but note that when the number of words in the vocabulary, the number of documents or the number of topics is large, the history can quickly overwhelm available active memory. Thus we recommend leaving the default of FALSE in most cases.

5. Changing Hidden Prior Defaults

In this section we overview some of the additional options that can be set using arguments included in the ... portion of the stm function. These functions change elements of the prior that are not likely to be necessary for the vast majority of users. The first section shows how to switch the topic prevalence prior to be sparsity promoting using the Gamma-Lasso penalty. Generally sparsity is not as important in this portion of the model but it can be useful in settings where there are a large number of variables with most expected to have no effect in topic prevalence. The next section describes how to set a penalty for the covariance matrix estimation. This can be useful if the topics are highly correlated.

5.1. Changing Estimation of Prevalence Covariate coefficients

The user can choose between two options "Pooled" is the default and estimates a model where the coefficients on topic prevalence have a zero-mean Normal prior with variance given a broad inverse-gamma hyperprior.

You can also choose "GL" which will use the Matt Taddy's gamlr package to estimate the coefficients using the Gamma-Lasso penalty. This penalty is sparsity inducing and will be more appropriate as the number of coefficients is larger relative to the number of documents, or when a lot of covariates are included to estimate topic prevalence.

```
> mod.out <- stm(documents, vocab, K=10,
+                prevalence= ~party, data=metadata,
+                max.em.its=2, gammamode="GL")
```

5.2. Changing Covariance Matrix Prior

The sigmaprior argument is a value between 0 and 1 defaulting to 0. The covariance between the topics Sigma is formed as a combination of the MLE of the full covariance matrix times (1-sigmaprior) plus the diagonal of the covariance matrix times (sigmaprior). Thus setting sigmaprior to 1 would result in a fully diagonal covariance matrix which in turn would make the topics weakly independent.[13]

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.

[13] There are also other priors that can be changed including prior on Tau taumode, tauprior, taumaxi. Changing the priors on tau affects the sparsity of the topic matrix.

References

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.

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

Gentzkow M, Shapiro JM (2010). "What drives media slant? Evidence from US daily newspapers." Econometrica, **78**(1), 35–71.

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

Roberts ME, Stewart BM, Airoldi EM (2013a). "A topic model for experimentation in the social sciences."

Roberts ME, Stewart BM, Tingley D, Airoldi EM (2013b). "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 (2014). "Structural topic models for open-ended survey responses." American Journal of Political Science.

Taddy M (2012). "Multinomial inverse regression for text analysis." Journal of the American Statistical Association, (just-accepted).

Affiliation:

Margaret E. Roberts
Department of Government
Harvard University
1737 Cambridge St, Cambridge, MA, USA
E-mail: roberts8@fas.harvard.edu
URL: http://scholar.harvard.edu/mroberts

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