Carnegie Mellon University

Text as Data

SICSS, May 13, 2025

Kristen Scotti

What is text analysis?

- Turning unstructured text into structured data for analysis
- Uses computational methods to extract patterns, meaning, or trends from text
- Combines linguistics, statistics, and programming

Examples:

- Analyzing social media sentiment
- Tracking topics in newspaper archives
- Identifying key themes in interview transcripts

Learning objectives

- Understand the basic steps of a text analysis pipeline
- Load, clean, and prepare textual data
- Apply simple natural language processing (NLP) techniques
- Visualize word patterns and frequency
- Think critically about interpretation and bias in text data

Text preprocessing

- Preprocessing = cleaning and preparing text before analysis
- Typical tasks:
    - Remove whitespace, punctuation, and HTML
    - Normalize text (e.g., lowercase)
    - Tokenize words
    - Remove stopwords

Why does it matter?

```
```{r}
duke_web_scrape <- "Duke Experts: A Trusted Source for Policy Makers\n\n\t\t\t\t\t\t\t"
```
```

Why does it matter?

```{r}
duke_web_scrape <- "Duke Experts: A Trusted Source for Policy Makers\n\n\t\t\t\t\t\t\t"
```

```{r}
gsub("\t", "", duke_web_scrape)
```

Why does it matter?

```{r}
duke_web_scrape <- "Duke Experts: A Trusted Source for Policy Makers\n\n\t\t\t\t\t\t"
```

```{r}
gsub("\t", "", duke_web_scrape)
```

```{r}
gsub("\t|\n", "", duke_web_scrape)
```

What is GREP?

- GREP = Globally search a Regular Expression and Print
- Used for pattern matching
- Functions: grep(), grepl()
  - grep()
    - returns the positions (indices) of matches in an input vector
  - grepl()
    - returns a logical vector (True/False)

GREP Example

- What do you get when you run this code?

```{r}
grepl("Experts", duke_web_scrape)
```

- Change “grepl” to “grep” – what do you get now?

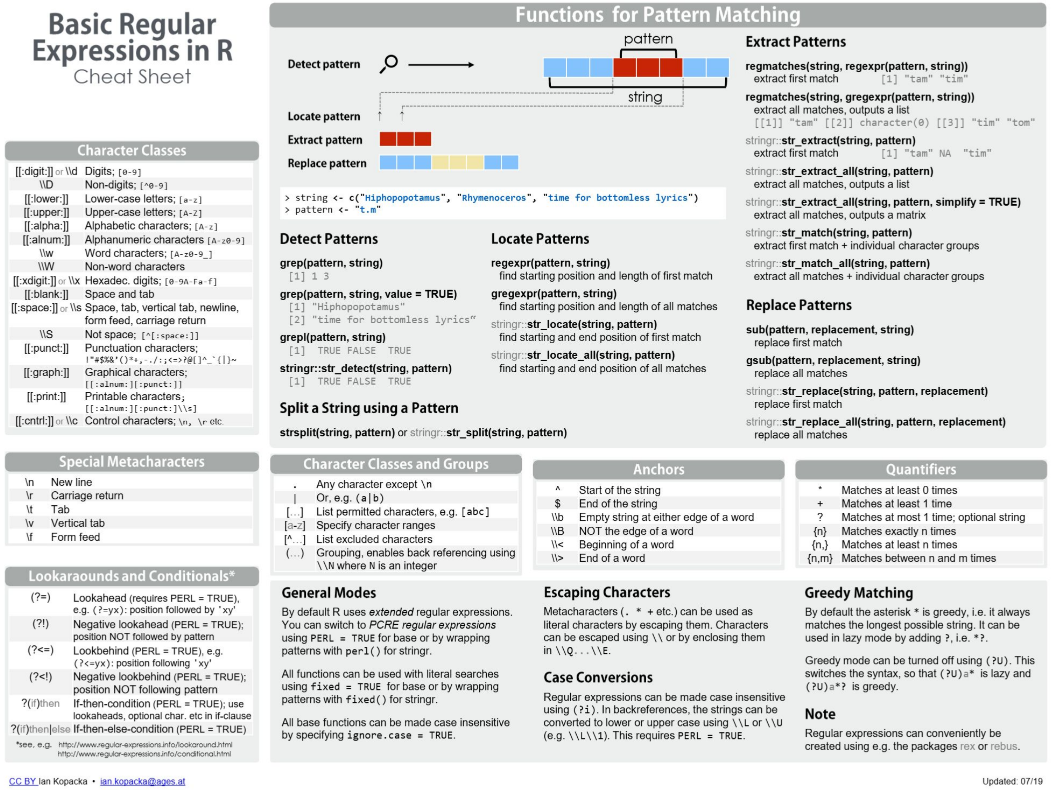
GREP Example

- What do you get when you run this code?

```
```{r}
some_text <- c("apple", "banana", "grape", "apricot")
grep("ap", some_text)
```
```

- Change “grepl” to “grep” – what do you get now?

GREP Example

- What do you get when you run this code?

```
```{r}
some_text<-c("Friends","don't","let","friends","make","wordclouds")
some_text[grep("^[F]", some_text)]
```
```

GREP Example - Special Characters

- What do you get when you run this code?

```
```{r}
text_chunk<-c("[R is FUN!]")
gsub("[","", text_chunk)
```
```

```
```{r}
text_chunk<-c("[R is FUN!]")
gsub('\\[|\\]',"", text_chunk)
```
```

Regex Cheat Sheet

- https://github.com/rstudio/cheatsheets/blob/main/regex.pdf



Basic Regular Expressions in R
Cheat Sheet

Character Classes

[[:digit:]] or \\d	Digits; [0-9]
\\D	Non-digits; [^0-9]
[[:lower:]]	Lower-case letters; [a-z]
[[:upper:]]	Upper-case letters; [A-Z]
[[:alpha:]]	Alphabetic characters; [A-z]
[[:alnum:]]	Alphanumeric characters [A-z0-9]
\\w	Word characters; [A-z0-9_]
\\W	Non-word characters
[[:xdigit:]] or \\x	Hexadec. digits; [0-9A-Fa-f]
[[:blank:]]	Space and tab
[[:space:]] or \\s	Space, tab, vertical tab, newline, form feed, carriage return
\\S	Not space; [^[:space:]]
[[:punct:]]	Punctuation characters; !"#$%&'()*+,-./:;<=>?@[]^_`{|}~
[[:graph:]]	Graphical characters; [[:alnum:][:punct:]]
[[:print:]]	Printable characters; [[:alnum:][:punct:]\\s]
[[:cntrl:]] or \\c	Control characters; \n, \r etc.

Special Metacharacters

\n	New line
\r	Carriage return
\t	Tab
\v	Vertical tab
\f	Form feed

Lookaraounds and Conditionals*

(?=)	Lookahead (requires PERL = TRUE), e.g. (?=yx): position followed by 'xy'
(?!)	Negative lookahead (PERL = TRUE); position NOT followed by pattern
(?<=)	Lookbehind (PERL = TRUE), e.g. (?<=yx): position following 'xy'
(?<!)	Negative lookbehind (PERL = TRUE); position NOT following pattern
?(if)then	If-then-condition (PERL = TRUE); use lookaheads, optional char. etc in if-clause
?(if)then|else	If-then-else-condition (PERL = TRUE)

*see, e.g. http://www.regular-expressions.info/lookaround.html
http://www.regular-expressions.info/conditional.html

Functions for Pattern Matching

Detect pattern
pattern
string
Locate pattern
Extract pattern
Replace pattern

```
> string <- c("Hiphopopotamus", "Rhymenoceros", "time for bottomless lyrics")
> pattern <- "t.m"
```

Detect Patterns

grep(pattern, string)
[1] 1 3
grep(pattern, string, value = TRUE)
[1] "Hiphopopotamus"
[2] "time for bottomless lyrics"
grepl(pattern, string)
[1] TRUE FALSE TRUE
stringr::str_detect(string, pattern)
[1] TRUE FALSE TRUE

Locate Patterns

regexpr(pattern, string)
find starting position and length of first match
gregexpr(pattern, string)
find starting position and length of all matches
stringr::**str_locate(string, pattern)**
find starting and end position of first match
stringr::**str_locate_all(string, pattern)**
find starting and end position of all matches

Split a String using a Pattern

strsplit(string, pattern) or stringr::**str_split(string, pattern)**

Extract Patterns

regmatches(string, regexpr(pattern, string))
extract first match [1] "tam" "tim"
regmatches(string, gregexpr(pattern, string))
extract all matches, outputs a list
[[1]] "tam" [[2]] character(0) [[3]] "tim" "tom"
stringr::**str_extract(string, pattern)**
extract first match [1] "tam" NA "tim"
stringr::**str_extract_all(string, pattern)**
extract all matches, outputs a list
stringr::**str_extract_all(string, pattern, simplify = TRUE)**
extract all matches, outputs a matrix
stringr::**str_match(string, pattern)**
extract first match + individual character groups
stringr::**str_match_all(string, pattern)**
extract all matches + individual character groups

Replace Patterns

sub(pattern, replacement, string)
replace first match
gsub(pattern, replacement, string)
replace all matches
stringr::**str_replace(string, pattern, replacement)**
replace first match
stringr::**str_replace_all(string, pattern, replacement)**
replace all matches

Character Classes and Groups

.	Any character except \n
|	Or, e.g. (a|b)
[...]	List permitted characters, e.g. [abc]
[a-z]	Specify character ranges
[^...]	List excluded characters
(...)	Grouping, enables back referencing using \\N where N is an integer

Anchors

^	Start of the string
$	End of the string
\\b	Empty string at either edge of a word
\\B	NOT the edge of a word
\\<	Beginning of a word
\\>	End of a word

Quantifiers

*	Matches at least 0 times
+	Matches at least 1 time
?	Matches at most 1 time; optional string
{n}	Matches exactly n times
{n,}	Matches at least n times
{n,m}	Matches between n and m times

General Modes

By default R uses *extended* regular expressions. You can switch to *PCRE regular expressions* using `PERL = TRUE` for base or by wrapping patterns with `perl()` for stringr.

All functions can be used with literal searches using `fixed = TRUE` for base or by wrapping patterns with `fixed()` for stringr.

All base functions can be made case insensitive by specifying `ignore.case = TRUE`.

Escaping Characters

Metacharacters (`. * +` etc.) can be used as literal characters by escaping them. Characters can be escaped using `\\` or by enclosing them in `\\Q...\\E`.

Case Conversions

Regular expressions can be made case insensitive using `(?i)`. In backreferences, the strings can be converted to lower or upper case using `\\L` or `\\U` (e.g. `\\L\\1`). This requires `PERL = TRUE`.

Greedy Matching

By default the asterisk `*` is greedy, i.e. it always matches the longest possible string. It can be used in lazy mode by adding `?`, i.e. `*?`.

Greedy mode can be turned off using `(?U)`. This switches the syntax, so that `(?U)a*` is lazy and `(?U)a*?` is greedy.

Note

Regular expressions can conveniently be created using e.g. the packages rex or rebus.

Updated: 07/19

What is tokenization?

- Tokenization = splitting text into meaningful units
  - Characters
  - Words (most common)
  - Sentences
  - N-grams: sequences of words of length n; for, "this is a sentence"
    - N = 1 (unigrams): this, is, a, sentence
    - N = 2 (bigrams): This is, is a, a sentence
    - N = 3 (trigrams): This is a, is a sentence
    - N-grams can be useful when word-order is important; nuances in sequences of words

Let's tokenize...

```{r}
sampleText <- "R is great for data analysis. It's also free!"
```

```{r}
data_frame(line = 1, text = sampleText) %>%
  unnest_tokens(word, text)
  #unnest_tokens(character, text, token = "characters")
  #unnest_tokens(sentence, text, token = "sentences")
  #unnest_tokens(bigram, text, token = "ngrams", n = 2)
  #unnest_tokens(trigram, text, token = "ngrams", n = 3)
```

line <dbl>	word <chr>
1	r
1	is
1	great
1	for
1	data
1	analysis
1	it's
1	also
1	free

Exercise

- Tokenize the sentence: "Learning R is fun and empowering!" into bigrams and trigrams.
- How does the meaning or context change based on how your unit of analysis?

Exercise

```{r}
# Write your code here!
text <- "Learning R is fun and empowering!"
tibble(line = 1, text = text) %>%
  unnest_tokens(word, text)

# Bigrams
unnest_tokens(tibble(line = 1, text = text), bigram, text, token = "ngrams", n = 2)
# Trigrams
unnest_tokens(tibble(line = 1, text = text), trigram, text, token = "ngrams", n = 3)


```

```
data.frame(line = 1, text = "Learning R is fun and empowering!") %>%
  unnest_tokens(word, text)
```

Exercise

"Learning R"
"R is"
"is fun"
"fun and"
"and empowering"

"Learning R is"
"R is fun"
"is fun and"
"fun and empowering"

How does unnest_tokens() handle punctuation or emojis?

- Punctuation marks are removed automatically (unless you customize the tokenizer)
- All text is converted to lowercase
- Emojis are kept if they are treated as characters or part of the Unicode word class, depending on the tokenization method

How does unnest_tokens() handle non-English characters?

- Tries to follow the Unicode standard, meaning:
- Accented characters (e.g., é, ñ, ü) and most Latin script characters are handled correctly
- Languages without spaces (like Chinese or Japanese) won't be tokenized correctly unless you use a custom tokenizer or a different package (e.g., jiebaR for Chinese or tokenizers.bpe for multilingual support).

What is a Corpus?

- A corpus is a collection of documents
- Tweets, articles, reviews, transcripts, ...

Load some data…

```{r}
original_data <- read.csv("Corona_NLP_train.csv") %>%
  select(created_at = TweetAt, text = OriginalTweet)

head(original_data)
```

After you load your data, tokenize it by word!

Tokenize

```{r}
# Write your code here!

tidy_covid <- original_data %>%
  unnest_tokens(word, text)

head(tidy_covid)
```

	created_at <chr>	word <chr>
1	3/16/2020	menyrbie
2	3/16/2020	phil_gahan
3	3/16/2020	chrisitv
4	3/16/2020	https
5	3/16/2020	t.co
6	3/16/2020	ifz9fan2pa

Counting

```{r}
tidy_covid %>%
  count(word) %>%
    arrange(desc(n))
```

word <chr>	n <int>
the	29744
to	25785
and	16344
https	15143
t.co	15132
of	14444
coronavirus	12929
a	12767
in	12556
for	9413

Remove stop words

```{r}
data("stop_words")

tidy_covid <- tidy_covid %>%
  anti_join(stop_words)

head(tidy_covid)
```

```{r}
tidy_covid %>%
  count(word) %>%
    arrange(desc(n))
```

word <chr>	n <int>
https	15143
t.co	15132
coronavirus	12929
19	7948
covid	7583
supermarket	4990
food	4963
prices	4942
store	4916
grocery	4340

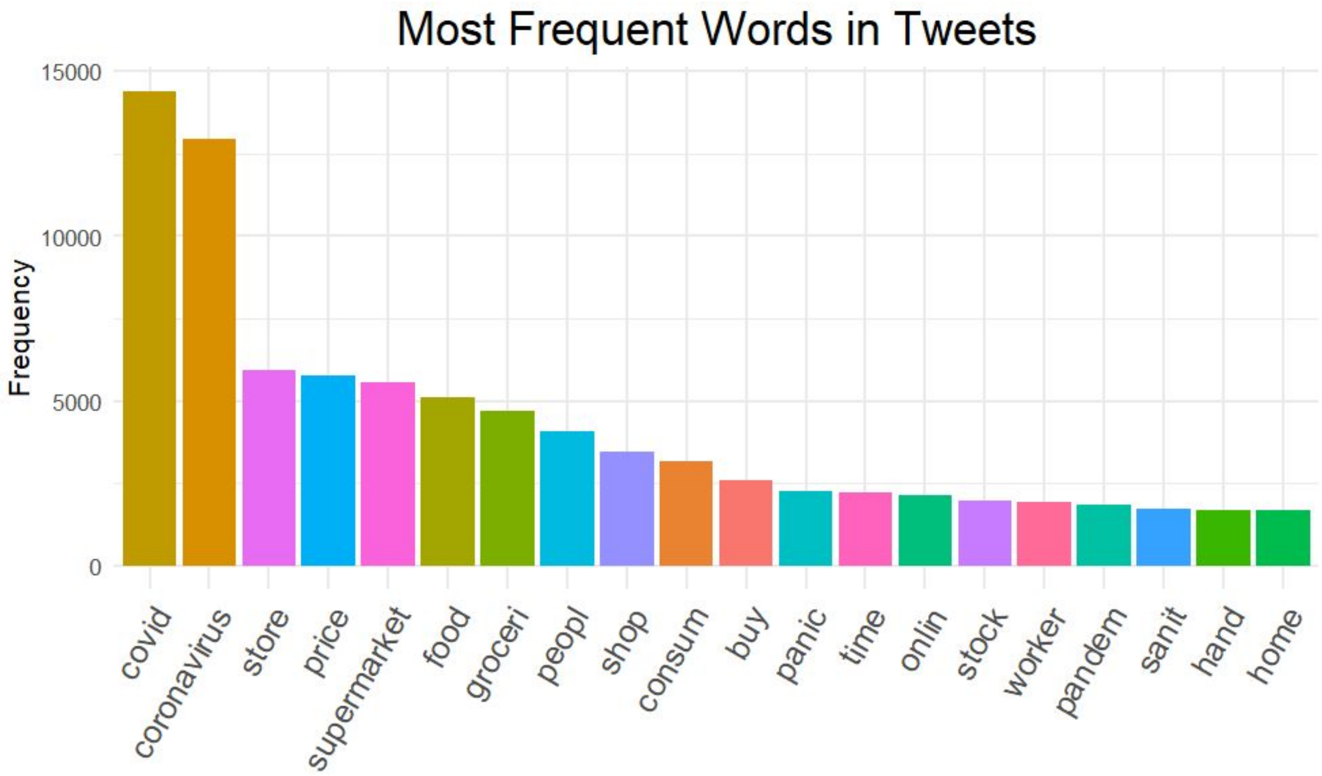
Remove Noise (Links, Twitter Junk)

```
```{r}
tidy_covid <- tidy_covid %>%
  filter(!word %in% c("https", "rt", "t.co", "amp"))
```
```

```
```{r}
tidy_covid %>%
  count(word) %>%
    arrange(desc(n))
```
```

word <chr>	n <int>
coronavirus	12929
19	7948
covid	7583
supermarket	4990
food	4963
prices	4942
store	4916
grocery	4340
people	4036
covid19	3192

Clean Punctuation, Numbers, and Whitespace

```
```{r}
tidy_covid$word <- gsub("[[:punct:]]", "", tidy_covid$word)
tidy_covid$word <- gsub("\\d+", "", tidy_covid$word)
tidy_covid <- tidy_covid %>% filter(word != "")
```
```

```
```{r}
tidy_covid %>%
  count(word) %>%
    arrange(desc(n))
```
```

word <chr>	n <int>
covid	14400
coronavirus	12939
supermarket	4990
food	4963
prices	4944
store	4917
grocery	4340
people	4036
consumer	2732
shopping	2294

Exercise: Clean a vector

```
c("Running!", "n95", "COVID-19", "great!!!", "1234")
```

Exercise: Clean a vector

word
<chr>
running
n
covid
great

c("Running!", "n95", "COVID-19", "great!!!", "1234")

````
```{r}
# Write your code here!
words <- tibble(word = c("Running!", "n95", "COVID-19", "great!!!", "1234")) %>%
  mutate(word = gsub("[[:punct:]]", "", word),
         word = gsub("\\d+", "", word),
         word = tolower(word))
```
````

Stemming

- Reduces words to their roots
- “typing” → “type”
- Use wordStem() from the SnowballC package

````
```{r}
tidy_covid <- tidy_covid %>%
  mutate(word = wordStem(word, language = "en"))
```
````

Word frequency analysis

````
```{r}
top_words <- tidy_covid %>%
  count(word) %>%
  arrange(desc(n))
```
````

Visualize...

```
```{r}
top_words %>%
  slice(1:20) %>%
    ggplot(aes(x=reorder(word, -n), y=n, fill=word))+
      geom_bar(stat="identity")+
        theme_minimal()+
        theme(axis.text.x =
            element_text(angle = 60, hjust = 1, size=13))+
        theme(plot.title =
            element_text(hjust = 0.5, size=18))+
          ylab("Frequency")+
          xlab("")+
          ggtitle("Most Frequent Words in Tweets")+
          guides(fill=FALSE)
```
```

Visualize…



Most Frequent Words in Tweets
Frequency
15000
10000
5000
0
covid
coronavirus
store
price
supermarket
food
groceri
peopl
shop
consum
buy
panic
time
onlin
stock
worker
pandem
sanit
hand
home

TF-IDF

- TF = Term Frequency
- IDF = Inverse Document Frequency
- Highlights words that are important in specific documents

TF-IDF

```{r}
tidy_covid_tfidf <- tidy_covid %>%
  count(created_at, word) %>%
  bind_tf_idf(word, created_at, n)
```

TF-IDF

Now let's see what the most unusual words are (highest TF-IDF score)

```{r}
top_tfidf <- tidy_covid_tfidf %>%
  arrange(desc(tf_idf))

top_tfidf$word[1:10]
```

```
 [1] "glasgow"              "stayathomeandstaysaf" "ajqfcjbj"
 [4] "apmacanada"           "bigmart"              "bzuivvzu"
 [7] "cmoldmak"             "cmqrsxnuv"            "cpykboekvt"
[10] "creativepapilo"
```

TF-IDF

Least distinctive word

```
```{r}
top_tfidf <- tidy_covid_tfidf %>%
  arrange(tf_idf)

# View the top tf-idf word
top_tfidf$word[1]
```
```

TF-IDF

Show top N most usual/unusual words with their TF-IDF score

```{r}
top_tfidf <- tidy_covid_tfidf %>%
  # arrange(desc(tf_idf)) %>%
  arrange(tf_idf) %>%
  select(word, tf_idf)

# View the top 10 words and their scores
head(top_tfidf, 10)
```

	word <chr>	tf_idf <dbl>
1	basic	0
2	call	0
3	chines	0
4	citi	0
5	come	0
6	communiti	0
7	consum	0
8	continu	0
9	coronavirus	0
10	covid	0

Exercise

Compare top TF-IDF terms by date

Exercise

Compare top TF-IDF terms by date

```{r}
data <- tibble(date = c("2020-01-01", "2020-01-01", "2020-01-02", "2020-01-02"),
               word = c("mask", "covid", "vaccine", "covid"))
data %>% count(date, word) %>%
  bind_tf_idf(word, date, n) %>%
  arrange(desc(tf_idf))
```

Exercise

Compare top TF-IDF terms by date

date <chr>	word <chr>	n <int>	tf <dbl>	idf <dbl>	tf_idf <dbl>
2020-01-01	mask	1	0.5	0.6931472	0.3465736
2020-01-02	vaccine	1	0.5	0.6931472	0.3465736
2020-01-01	covid	1	0.5	0.0000000	0.0000000
2020-01-02	covid	1	0.5	0.0000000	0.0000000

Sentiment Analysis

- Use get_sentiments("bing")
- Join with tokenized text
- Count sentiment by date

```{r}
head(get_sentiments("bing"))
```

A tibble: 6 x 2

word <chr>	sentiment <chr>
2-faces	negative
abnormal	negative
abolish	negative
abominable	negative
abominably	negative
abominate	negative

6 rows

Apply the Bing sentiment dictionary to our dataset

```
```{r}
covid_sentiment <- tidy_covid %>%
  inner_join(get_sentiments("bing")) %>%
    count(created_at, sentiment)

head(covid_sentiment)
```
```

```
```{r}
tidy_covid$date <- as.Date(tidy_covid$created_at, format = "%m/%d/%Y")
```
```

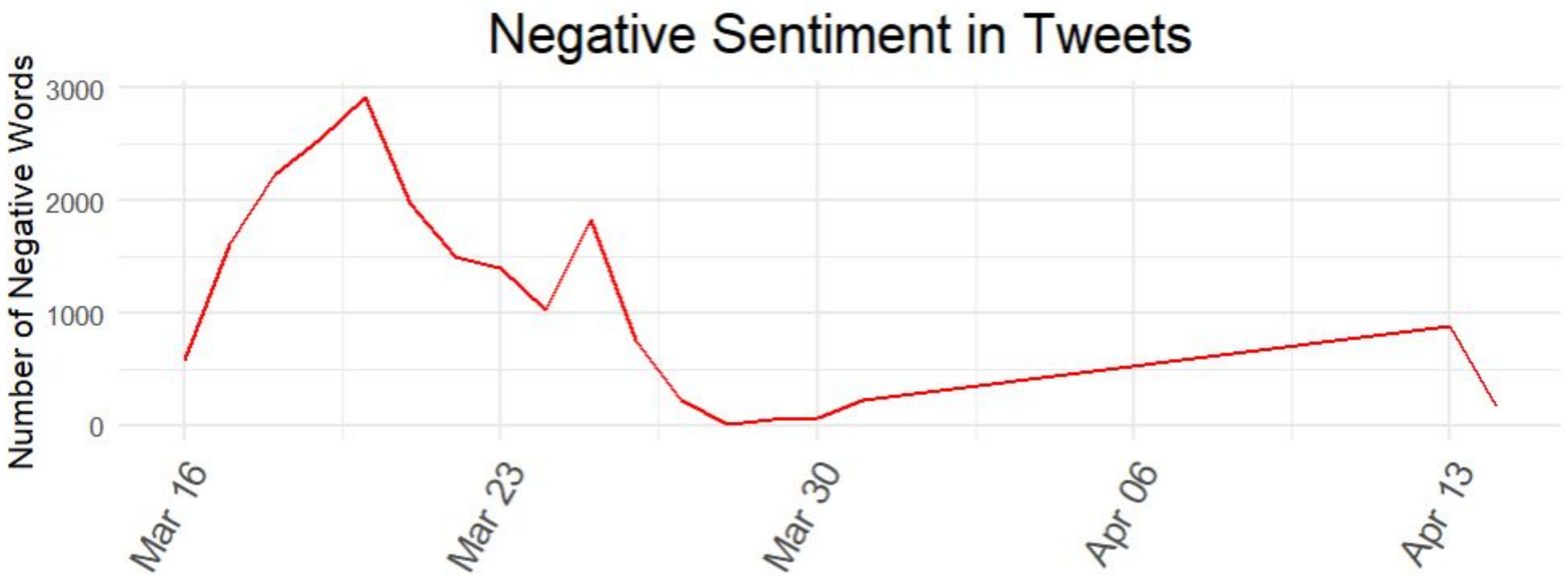
Apply the Bing sentiment dictionary to our dataset

```{r}
covid_sentiment_plot <-
  tidy_covid %>%
    inner_join(get_sentiments("bing")) %>%
      filter(sentiment=="negative") %>%
          count(date, sentiment)
```

Plot

```{r}
ggplot(covid_sentiment_plot, aes(x=date, y=n))+
  geom_line(color="red", size=.5)+
    theme_minimal()+
    theme(axis.text.x =
            element_text(angle = 60, hjust = 1, size=13))+
    theme(plot.title =
            element_text(hjust = 0.5, size=18))+
      ylab("Number of Negative Words")+
      xlab("")+
      ggtitle("Negative Sentiment in Tweets")+
      theme(aspect.ratio=1/4)
```

Plot


Negative Sentiment in Tweets
Number of Negative Words
3000
2000
1000
0
Mar 16
Mar 23
Mar 30
Apr 06
Apr 13

Questions…

- Can one tweet have both positive and negative sentiment?
- What if a word isn’t in the lexicon?

Dictionary-Based Analysis

```{r}
economic_dictionary <- c("economy", "unemployment", "trade", "tariffs", "shopping")
```

Dictionary-Based Analysis

Now use stringr::str_detect() to filter tweets that mention any of these terms:

```
```{r}
economic_tweets <- original_data %>%
  filter(str_detect(text, paste(economic_dictionary, collapse = "|")))

head(economic_tweets)
```
```

Dictionary-Based Analysis

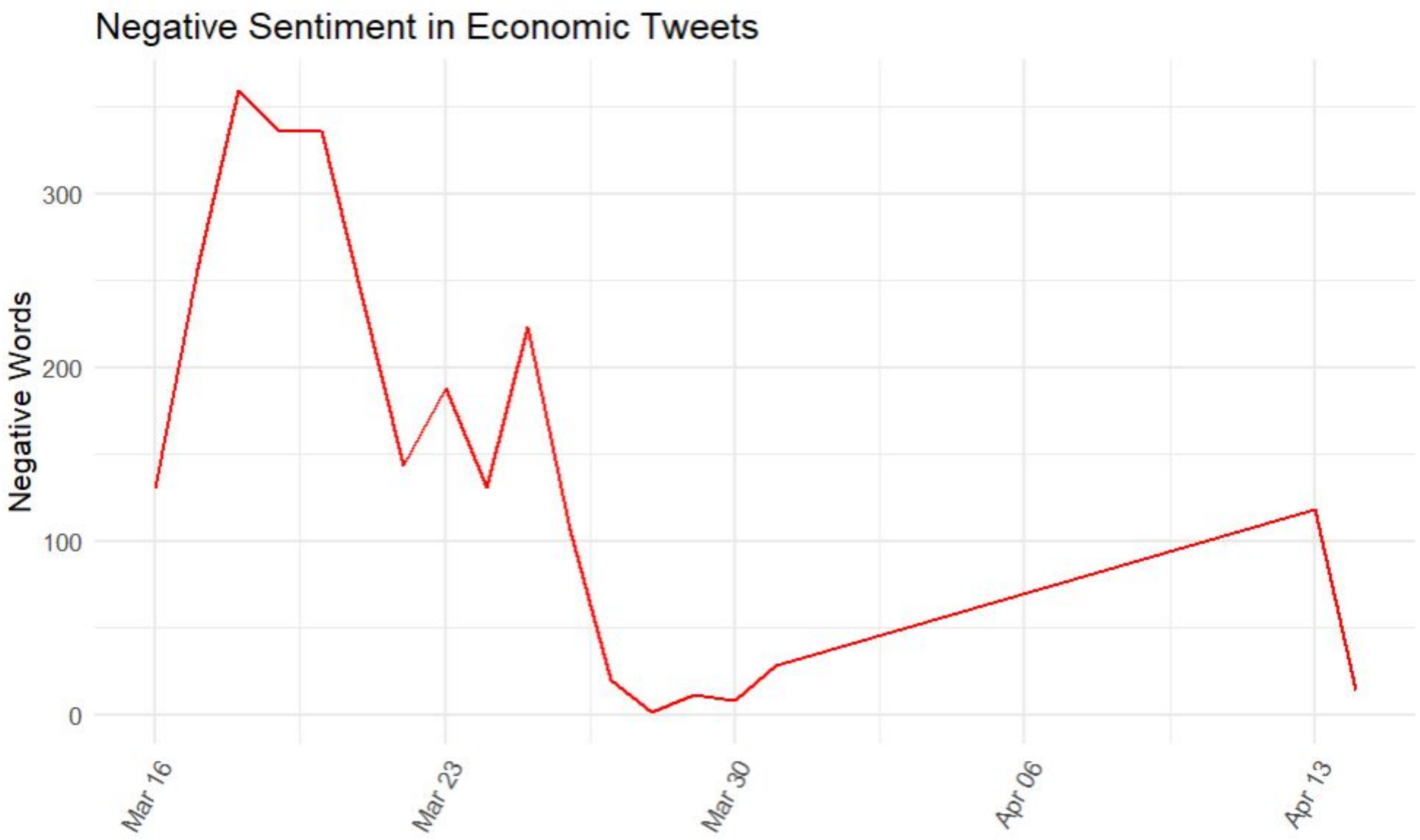
```{r}
# Filter original tweets to only those that match your economic dictionary
economic_tweets <- original_data %>%
  filter(str_detect(text, paste(economic_dictionary, collapse = "|")))

# Then tokenize only these tweets
tidy_economic <- economic_tweets %>%
  unnest_tokens(word, text)

# Now join to Bing and analyze sentiment
economic_sentiment <- tidy_economic %>%
  inner_join(get_sentiments("bing")) %>%
  count(created_at, sentiment)

# Optionally, plot only negative sentiment
economic_sentiment_plot <- economic_sentiment %>%
  filter(sentiment == "negative") %>%
  mutate(date = as.Date(created_at, format = "%m/%d/%Y"))  # fix format if needed

ggplot(economic_sentiment_plot, aes(x = date, y = n)) +
  geom_line(color = "red", linewidth = 0.5) +
  theme_minimal() +
  labs(x = "", y = "Negative Words", title = "Negative Sentiment in Economic Tweets") +
  theme(axis.text.x = element_text(angle = 60, hjust = 1))
```



Negative Sentiment in Economic Tweets
Negative Words
0
100
200
300
Mar 16
Mar 23
Mar 30
Apr 06
Apr 13

Exercise: Create your own dictionary

- e.g., healthcare, misinformation, etc.
- Filter or visualize sentiment over time
- Compare peaks in negative/positive sentiment

Topic Modeling

- A method to discover hidden themes ("topics") in large collections of text
- Assumes:
  - Each document is a mixture of topics
  - Each topic is a mixture of words
- Helps explore, summarize, or classify text data
- Latent Dirichlet Allocation (LDA)
- Structural Topic Modeling (STM)

```
data("AssociatedPress")
```

LDA on Associated Press Data

- Built-in dataset: AssociatedPress → DocumentTermMatrix
    - Rows = documents
    - Columns = terms/words
    - Values = how many times each word appears in each document
- Fit model with 10 topics)


```
data("AssociatedPress")
```


```
# Fits a Latent Dirichlet Allocation (LDA) model with 10 topics on the AP dataset.
AP_topic_model<-LDA(AssociatedPress, k=10, control = list(seed = 321))
```

```r
data("AssociatedPress")
```

LDA on Associated Press Data

```r
# Fits a Latent Dirichlet Allocation (LDA) model with 10 topics on the AP dataset.
AP_topic_model<-LDA(AssociatedPress, k=10, control = list(seed = 321))
```

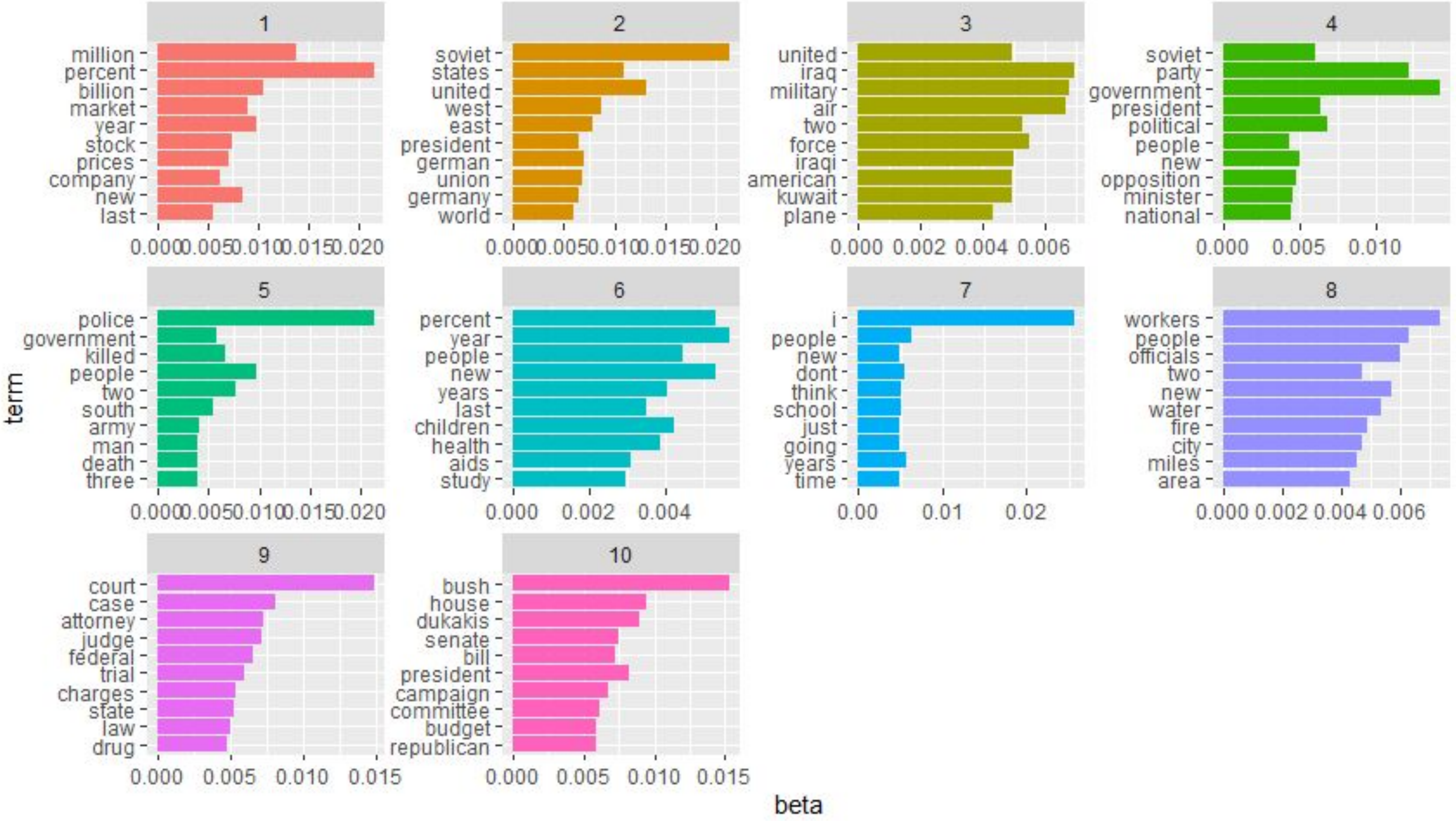
Extract and Visualize Top Terms

```r
AP_topics <- tidy(AP_topic_model, matrix = "beta")
```

```r
# Finds the top 10 terms for each topic, based on the highest beta values (topic-word probabilities).
ap_top_terms <- AP_topics %>%
  group_by(topic) %>%
  top_n(10, beta) %>%
  ungroup() %>%
  arrange(topic, -beta)
```

Extract and Visualize Top Terms

```
ap_top_terms %>%
  mutate(term = reorder(term, beta)) %>%
  ggplot(aes(term, beta, fill = factor(topic))) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~ topic, scales = "free") +
  coord_flip()
```


1
million
percent
billion
market
year
stock
prices
company
new
last
2
soviet
states
united
west
east
president
german
union
germany
world
3
united
iraq
military
air
two
force
iraqi
american
kuwait
plane
0.000 0.002 0.004 0.006
4
soviet
party
government
president
political
people
new
opposition
minister
national
0.000 0.005 0.010
5
police
government
killed
people
two
south
army
man
death
three
6
percent
year
people
new
years
last
children
health
aids
study
0.000 0.002 0.004
7
i
people
new
dont
think
school
just
going
years
time
0.00 0.01 0.02
8
workers
people
officials
two
new
water
fire
city
miles
area
0.000 0.002 0.004 0.006
9
court
case
attorney
judge
federal
trial
charges
state
law
drug
0.000 0.005 0.010 0.015
10
bush
house
dukakis
senate
bill
president
campaign
committee
budget
republican
0.000 0.005 0.010 0.015
term
beta

STM example on Poliblogs

- Dataset: 2008 political blog posts
- Preprocess with textProcessor()
- Format with prepDocuments()

```
google_doc_id <- "1LcX-JnpGB0lU1iDnXnxB6WFqBywUKpew" # google file ID
poliblogs<-read.csv(sprintf("https://docs.google.com/uc?id=%s&export=download",
                            google_doc_id), stringsAsFactors = FALSE)
```

Preprocessing

```
# Preprocess and prepare documents
processed <- textProcessor(poliblogs$documents, metadata = poliblogs)
out <- prepDocuments(processed$documents, processed$vocab, processed$meta)
docs <- out$documents
vocab <- out$vocab
meta <- out$meta
```

Document: A single text entry, like a blog post.
Vocabulary: The list of all words that remain after cleaning.
Metadata: Additional info about each document, such as date or political affiliation.

What is a covariate?

- A covariate is extra information about a document that might explain why certain topics appear more often
- A blog's political rating might influence which topics are more common.
- The date might explain how topics shift over time.

What is K?

- K is the number of topics to extract
- Chosen by YOU — not learned from the data
- Too small → overly broad topics
- Too large → fragmented or incoherent topics
- Use searchK() to evaluate values for K

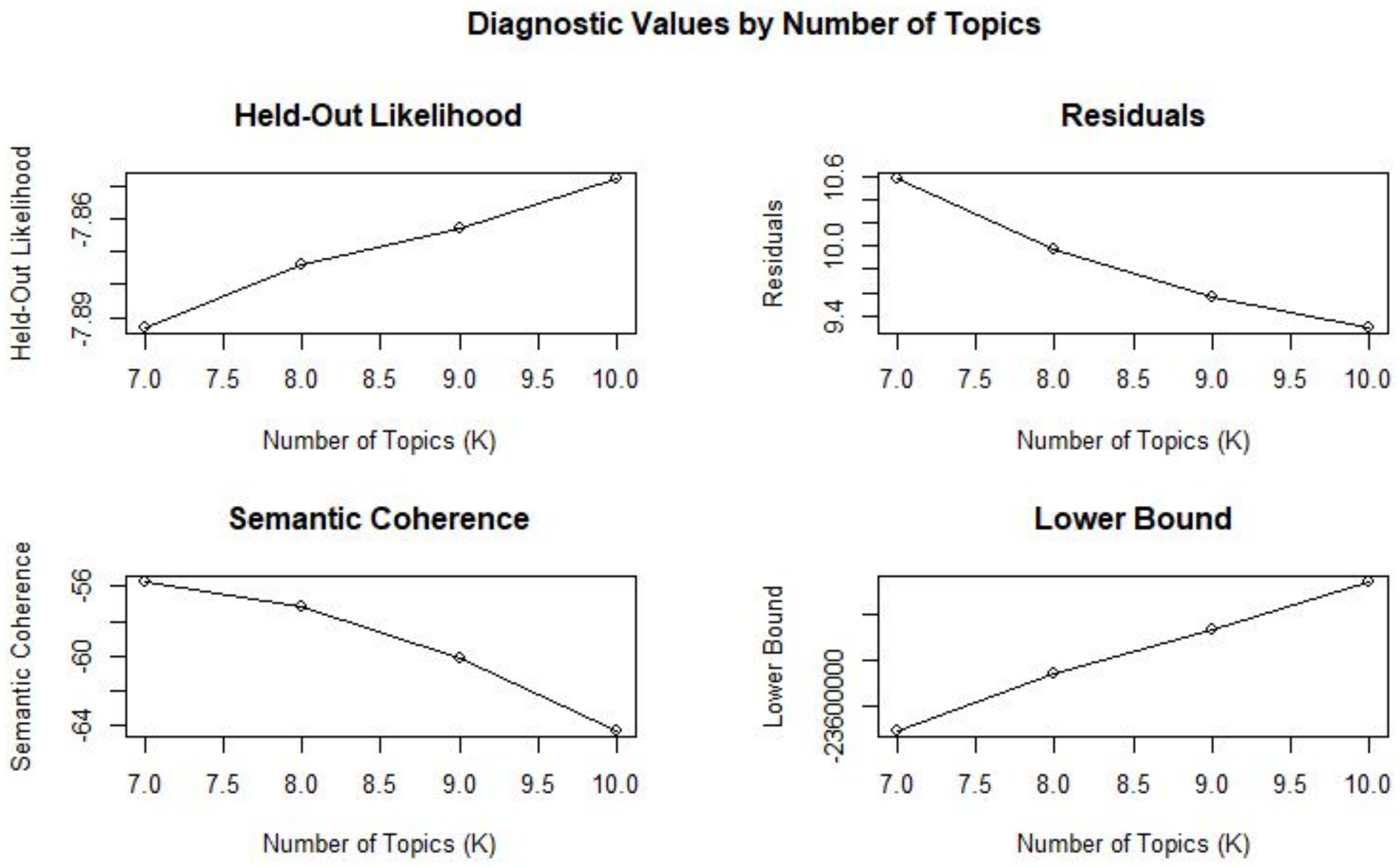
Finding k

```r
# ----------------------------
# OPTION A: Run and Save searchK
# ----------------------------
# Warning: This process is computationally intensive,
# Search for Optimal Number of Topics (K)
findingk <- searchK(
  documents = docs,
  vocab = vocab,
  K = 7:10,
  prevalence = ~ rating + s(day),
  data = meta,
  verbose = FALSE
)
save(findingk, file = "findingk.Rda")
```

```r
# ----------------------------
# OPTION B: Load precomputed searchK
# ----------------------------
load("findingk.Rda")
plot(findingk)
```

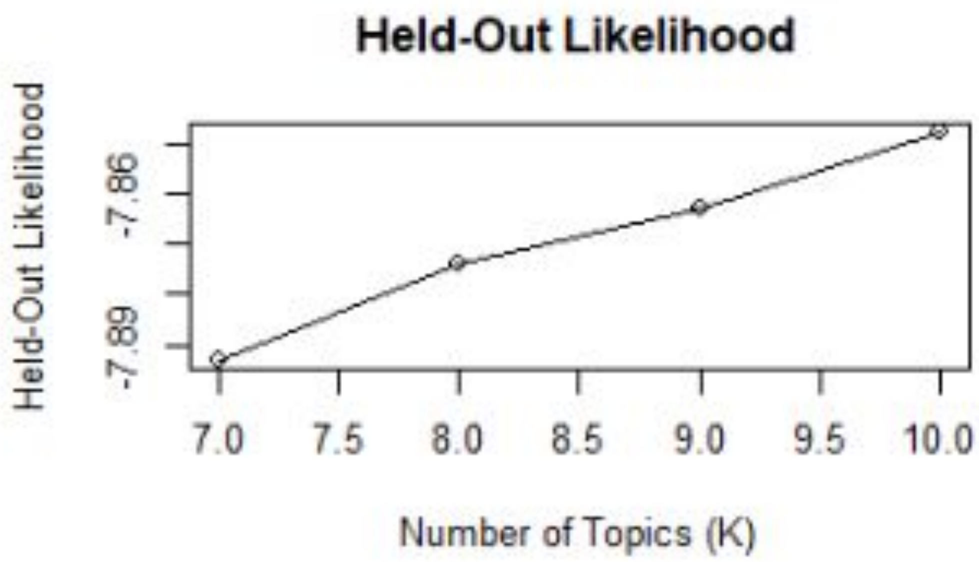
prevalence = ~ rating + s(day) includes covariates:

- rating: Political leaning
- s(day): A spline, which models non-linear change over time


Diagnostic Values by Number of Topics
Held-Out Likelihood
Held-Out Likelihood
-7.86
-7.89
Number of Topics (K)
7.0 7.5 8.0 8.5 9.0 9.5 10.0
Residuals
Residuals
10.6
10.0
9.4
Number of Topics (K)
7.0 7.5 8.0 8.5 9.0 9.5 10.0
Semantic Coherence
Semantic Coherence
-56
-60
-64
Number of Topics (K)
7.0 7.5 8.0 8.5 9.0 9.5 10.0
Lower Bound
Lower Bound
-2360000
Number of Topics (K)
7.0 7.5 8.0 8.5 9.0 9.5 10.0

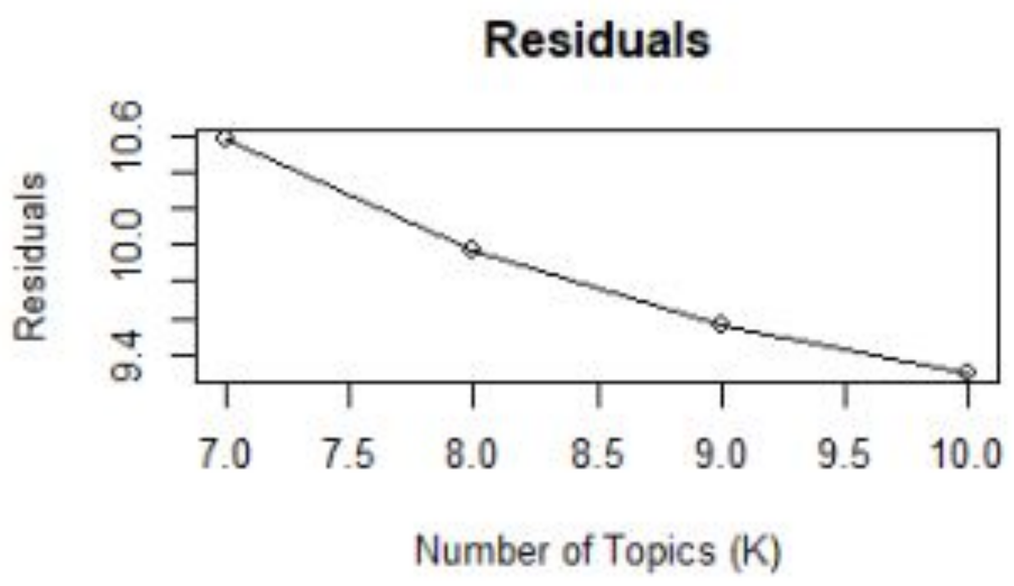
Held-out likelihood

- Higher is better
- Increases steadily → 10 is best here



Residuals

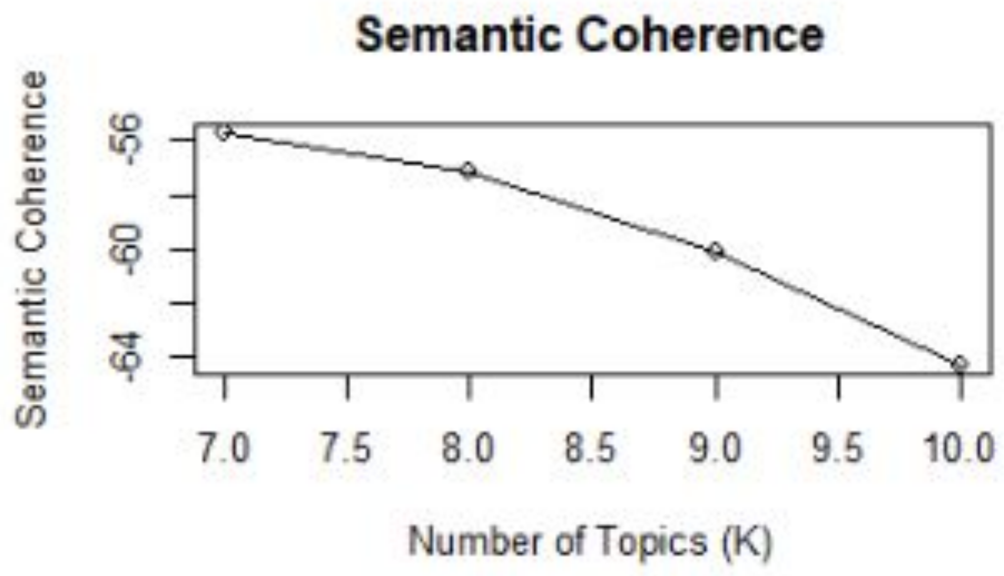
- Higher is better
- Increases steadily → 10 is best here



Carnegie Mellon University

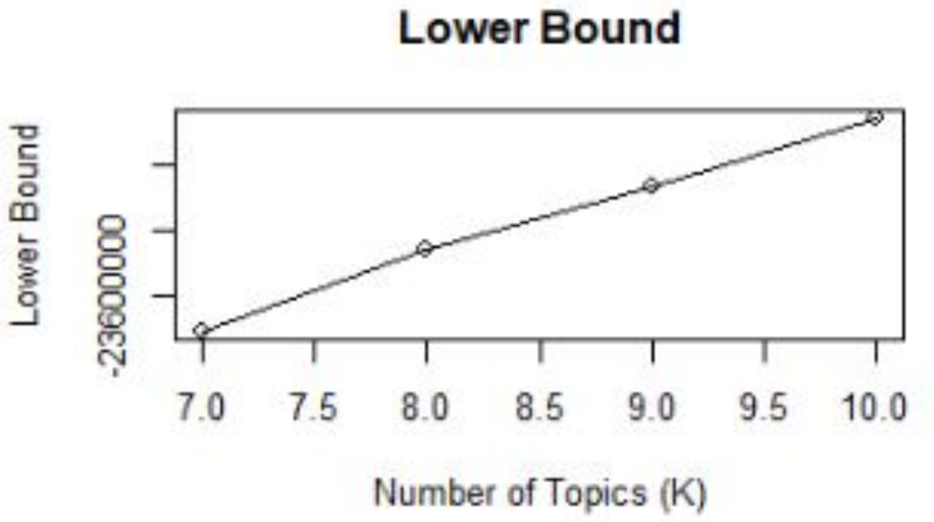
Semantic Coherence

- Higher is better
- Decreases as K increases
  → 7 is best here



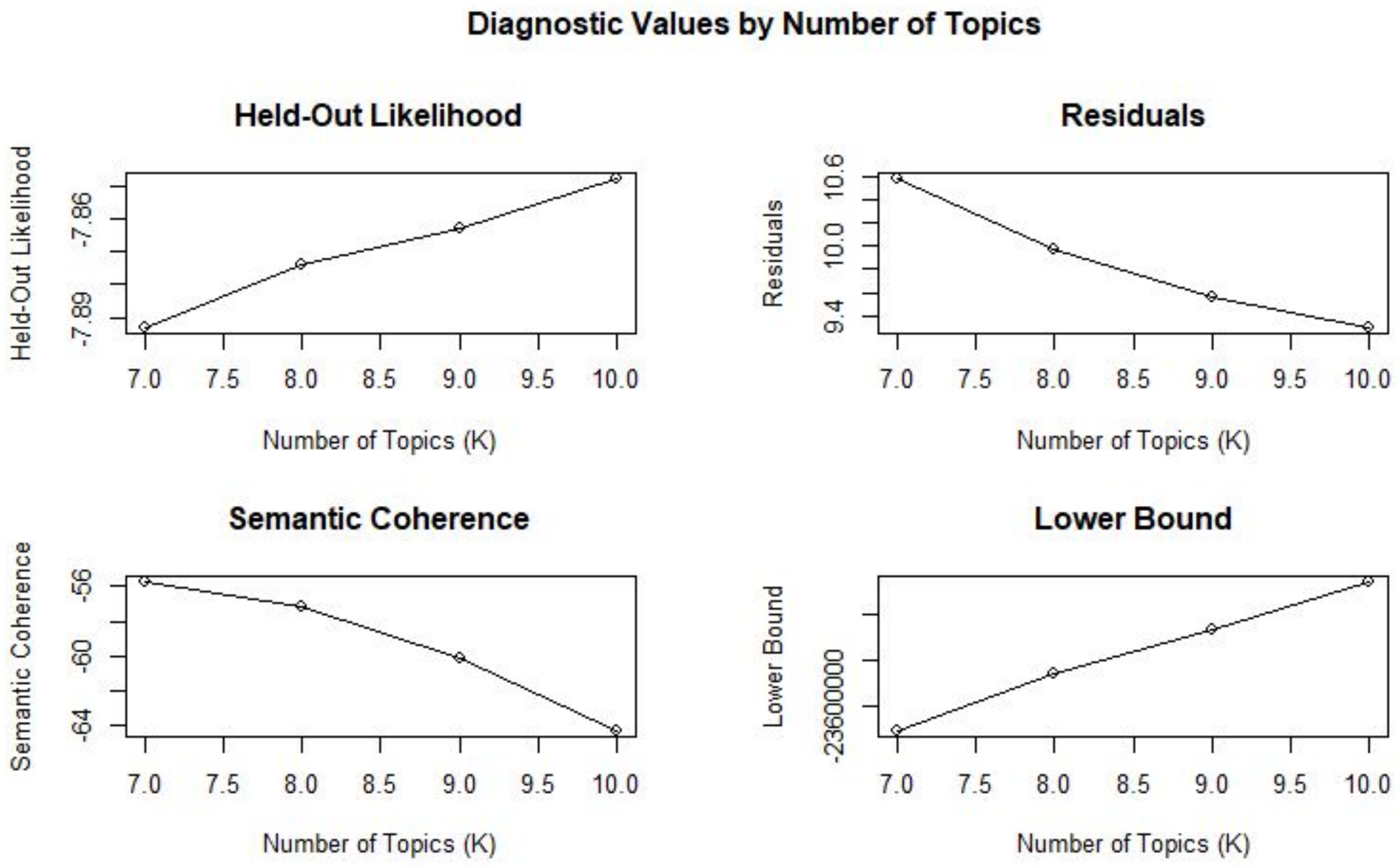
Lower bound

- Less negative is better
- Improves with K, but used mainly for convergence



Trade-offs

- Held-Out Likelihood and Residuals both point to K = 10 as the most statistically powerful model—it generalizes best to new data and captures more structure.
- Semantic Coherence drops off significantly after K = 7, which suggests that topics become less interpretable—i.e., the top words in each topic co-occur less often in documents

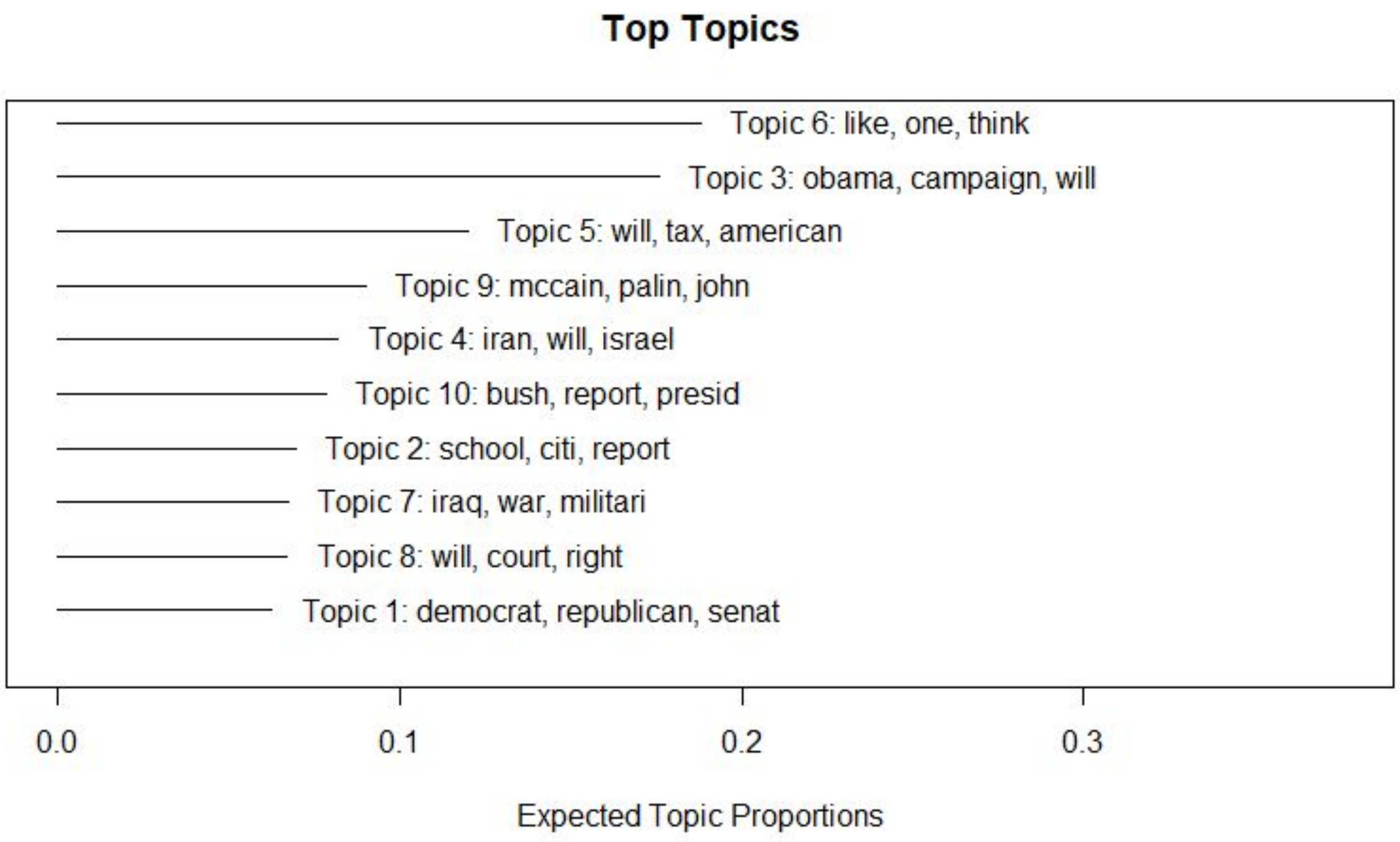

Diagnostic Values by Number of Topics
Held-Out Likelihood
Held-Out Likelihood
-7.86
-7.89
7.0
7.5
8.0
8.5
9.0
9.5
10.0
Number of Topics (K)
Residuals
Residuals
10.6
10.0
9.4
Number of Topics (K)
Semantic Coherence
Semantic Coherence
-56
-60
-64
Number of Topics (K)
Lower Bound
Lower Bound
-2360000
Number of Topics (K)

STM – Fit

```
# ----------------------------
# Fit STM with chosen K (e.g., 10)
# ----------------------------
First_STM <- stm(
  documents = docs,
  vocab = vocab,
  K = 10,  # use selected K from searchK
  prevalence = ~ rating + s(day),
  max.em.its = 75,
  data = meta,
  init.type = "Spectral",
  verbose = FALSE
)

plot(First_STM)
```

Top topics



Top Topics
Topic 6: like, one, think
Topic 3: obama, campaign, will
Topic 5: will, tax, american
Topic 9: mccain, palin, john
Topic 4: iran, will, israel
Topic 10: bush, report, presid
Topic 2: school, citi, report
Topic 7: iraq, war, militari
Topic 8: will, court, right
Topic 1: democrat, republican, senat
0.0
0.1
0.2
0.3
Expected Topic Proportions

Look at example texts for a topic

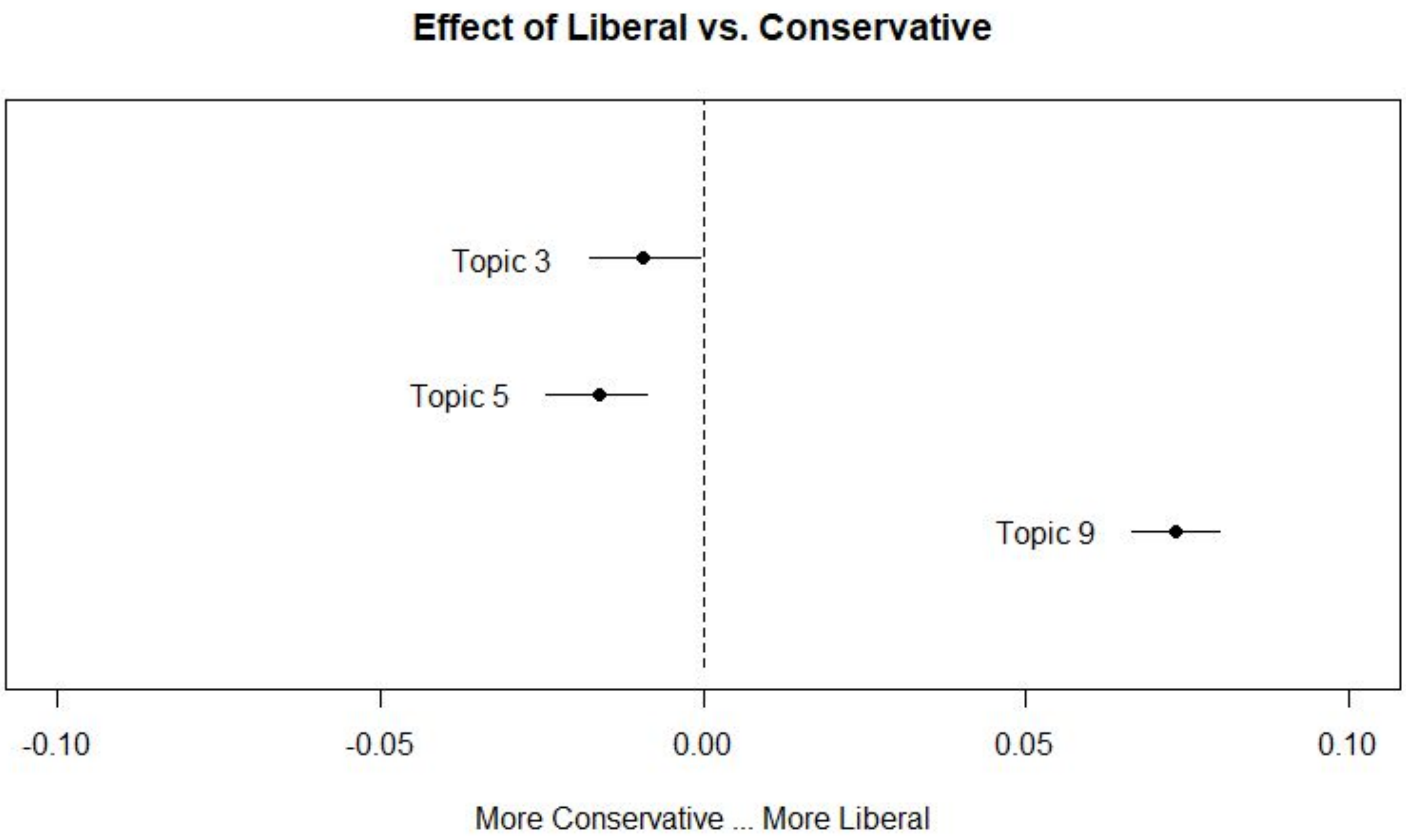
```
# ----------------------------
# Examine example topics
# ---------------------------
findThoughts(First_STM, texts = poliblogs$documents,
             n = 2, topics = 3)
```

Estimate and plot topic effects

```
# ----------------------------
# Estimate and plot topic effects by covariate
# ----------------------------
predict_topics<-estimateEffect(formula = 1:10 ~ rating + s(day), stmobj = First_STM,
                               metadata = out$meta, uncertainty = "Global")
```

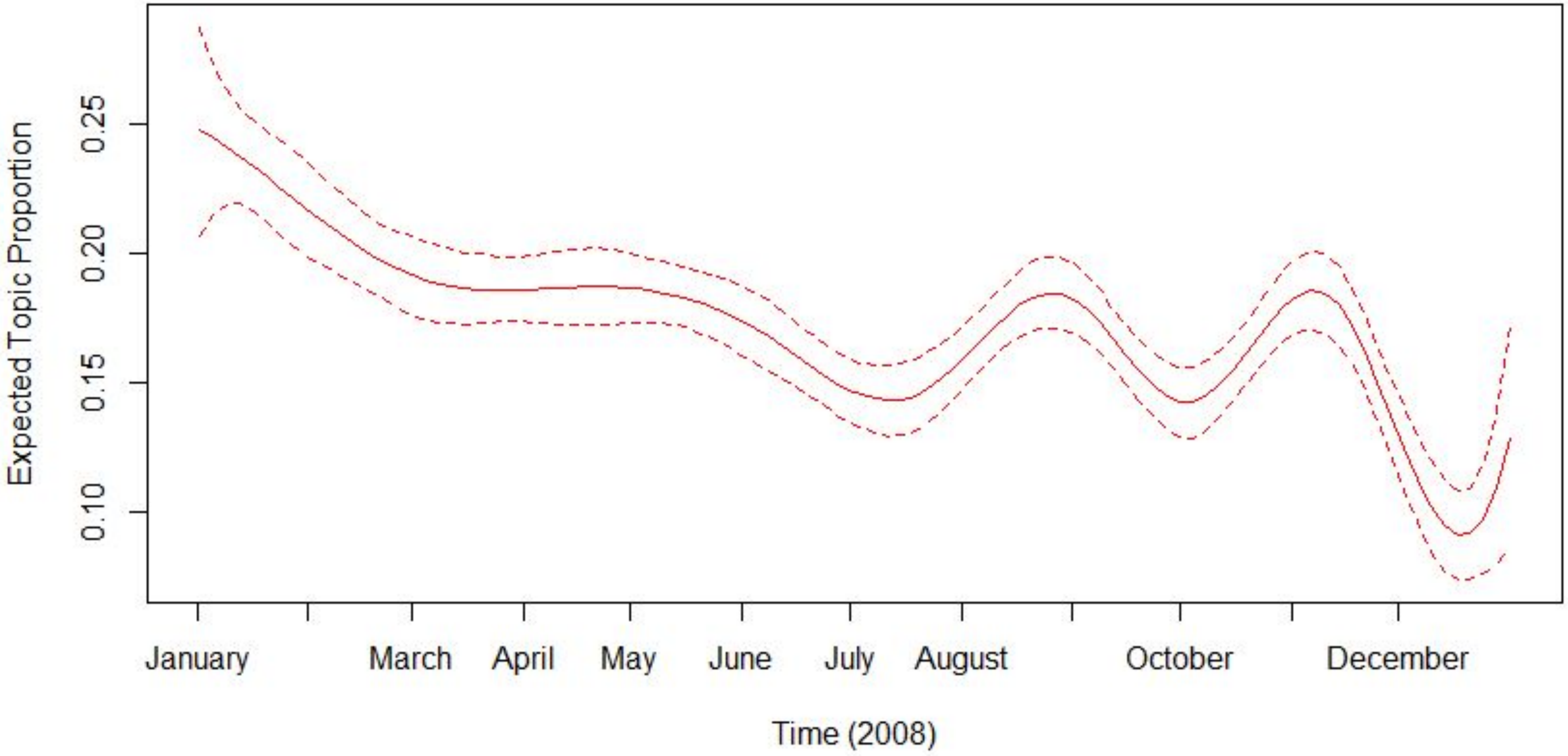
Visualize Covariate Effects

```
# Liberal vs. Conservative
plot(predict_topics, covariate = "rating", topics = c(3, 5, 9),
     model = First_STM, 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('Topic 3', 'Topic 5', 'Topic 9'))
```


Effect of Liberal vs. Conservative
Topic 3
Topic 5
Topic 9
-0.10
-0.05
0.00
0.05
0.10
More Conservative ... More Liberal

```r
# Change over time
plot(predict_topics, "day", method = "continuous", topics = 3,
     model = First_STM, 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)
```


Expected Topic Proportion
0.10
0.15
0.20
0.25
January
March
April
May
June
July
August
October
December
Time (2008)

Brainstorming time...

- **Topic or Research Question**
  - What kind of text would you analyze? What do you want to know?
- **Approach/Tools**
  - Would you use sentiment analysis? TF-IDF? Topic modeling? Something else?
- **Potential Pitfalls**
  - What challenges might come up with the data, methods, or interpretation?
  - Are there any risks of bias or misinterpretation in your data, methods, or results? e.g., biased training data, missing context, or over-reliance on sentiment dictionaries