

Text Mining Methodologies with R: An Application to Central Bank Texts*

Jonathan Benchimol,[†] Sophia Kazinnik[‡] and Yossi Saadon[§]

December 10, 2020

We review several existing methodologies in text analysis and explain formal processes of text analysis using the open-source software R and relevant packages. We present some technical applications of text mining methodologies comprehensively to economists.

1 Introduction

A large and growing amount of unstructured data is available nowadays. Most of this information is text-heavy, including articles, blog posts, tweets and more formal documents (generally in Adobe PDF or Microsoft Word formats). This availability presents new opportunities for researchers, as well as new challenges for institutions. In this paper, we review several existing methodologies in analyzing text and describe a formal process of text analytics using open-source software R. Besides, we discuss potential empirical applications.

This paper is a primer on how to systematically extract quantitative information from unstructured or semi-structured data (texts). Text mining, the quantitative representation of text, has been widely used in disciplines such as political science, media, and security. However, an emerging body of literature began to apply it to the analysis of macroeconomic issues, studying central bank commu-

*This paper does not necessarily reflect the views of the Bank of Israel, the Federal Reserve Bank of Richmond or the Federal Reserve System. The present paper serves as the technical appendix of our research paper (Benchimol et al., 2020). We thank Itamar Caspi, Shir Kamenetsky Yadan, Ariel Mansura, Ben Schreiber, and Bar Weinstein for their productive comments.

[†]Bank of Israel, Jerusalem, Israel. Corresponding author. Email: jonathan.benchimol@boi.org.il

[‡]Federal Reserve Bank of Richmond, Richmond, VA, USA.

[§]Bank of Israel, Jerusalem, Israel.

nication and financial stability in particular.¹ This type of text analysis is gaining popularity, and is becoming more widespread through the development of technical tools facilitating information retrieval and analysis.²

An applied approach to text analysis can be described by several sequential steps. A uniform approach to creating such measurement is required to assign a quantitative measure to this type of data. To quantify and compare texts, they need to be measured uniformly. Roughly, this process can be divided into four steps. These steps include data selection, data cleaning process, extraction of relevant information, and its subsequent analysis.

We describe briefly each step below and demonstrate how it can be executed and implemented using open source R software. We use a set of monthly reports published by the Bank of Israel as our data set.

Several applications are possible. An automatic and precise understanding of financial texts could allow for the construction of several financial stability indicators. Central bank publications (interest rate announcements, official reports, etc.) could also be analyzed. This quick and automatic analysis of the sentiment conveyed by these texts should allow for fine-tuning of these publications before making them public. For instance, a spokesperson could use this tool to analyze the orientation of a text—an interest rate announcement for example—before making it public.

The remainder of the paper is organized as follows. Section 2 describes text extraction and Section presents 3 methodologies for cleaning and storing text for text mining. Section 4 presents several data structures used in Section 5 which details methodologies used for text analysis. Section 6 concludes, and the Appendix presents additional results.

¹See, for instance, Bholat et al. (2015), Bruno (2017), and Correa et al. (2020).

²See, for instance, Lexalytics, IBM Watson AlchemyAPI, Provalis Research Text Analytics Software, SAS Text Miner, Sysomos, Expert System, RapidMiner Text Mining Extension, Clarabridge, Luminoso, Bitext, Etuma, Synapsify, Medallia, Abzooba, General Sentiment, Semantria, Kanjoya, Twinword, VisualText, SIFT, Buzzlogix, Averbis, AYLIEN, Brainspace, OdinText, Loop Cognitive Computing Appliance, ai-one, LingPipe, Megaputer, Taste Analytics, LinguaSys, muText, TextualETL, Ascribe, STATISTICA Text Miner, MeaningCloud, Oracle Endeca Information Discovery, Basis Technology, Language Computer, NetOwl, DiscoverText, Angoos KnowledgeREADER, Forest Rim's Textual ETL, Pingar, IBM SPSS Text Analytics, OpenText, Smartlogic, Narrative Science Quill, Google Cloud Natural Language API, TheySay, indico, Microsoft Azure Text Analytics API, Datumbox, Relativity Analytics, Oracle Social Cloud, Thomson Reuters Open Calais, Verint Systems, Intellexer, Rocket Text Analytics, SAP HANA Text Analytics, AUTINDEX, Text2data, Saplo, and SYSTRAN, among many others.

2 Text extraction

Once a set of texts is selected, it can be used as an input using package `tm` (Feinerer et al., 2008) within the open-source software R. This package can be thought as a framework for text mining applications within R, including text preprocessing.

This package has a function called `Corpus`. This function takes a predefined directory which contains the input (a set of documents) and returns the output, which is the set of documents organized in a particular way. In this paper, we refer to this output as a corpus. `Corpus` here is a framework for storing this set of documents.

We define our corpus through R in the following way. First, we apply a function called `file.path`, that defines a directory where all of our text documents are stored.³ In our example, it is the folder that stores all 220 text documents, each corresponding to a separate interest rate decision meeting.

After we define the working directory, we apply the function `Corpus` from the package `tm` to all of the files in the working directory. This function formats the set of text documents into a corpus object class as defined internally by the `tm` package.

```
file.path <- file.path("../data/folder")
corpus <- Corpus(DirSource(file.path))
```

Now, we have our documents organized as a corpus object. The content of each document can be accessed and read using the `writeLines` function. For example:

```
writeLines(as.character(corpus[[1]]))
```

Using this command line, we can access and view the content of document number one within the corpus. This document corresponds to the interest rate discussion published in December 2007. Below are the first two sentences of this document:

```
Bank of Israel Report to the public of the Bank of Israel's discussions
before setting the interest rate for January 2007 The broad-forum
discussions took place on December 24, 2006, and the narrow forum
```

³The folder should contain text documents only. If there are other files in that location (i.e. R files), then the `Corpus` function will include the text in the other files.

discussions on December 25, 2006, January 2007 General Before the Governor makes the monthly interest rate decision, discussions are held at two levels. The first discussion takes place in a broad forum, in which the relevant background economic conditions are presented, including real and monetary developments in Israel's economy and developments in the global economy.

There are other approaches to storing a set of texts in R, for example by using the function `data.frame` or `tibble`, however, we will concentrate on `tm`'s corpus approach as it is more intuitive, and has a greater number of corresponding functions explicitly written for text analysis.

3 Cleaning and storing text

Once the relevant corpus is defined, we transform it into an appropriate format for further analysis. As mentioned previously, each document can be thought of as a set of tokens. Tokens are sets of words, numbers, punctuation marks, and any other symbols present in the given document. The first step of any text analysis framework is to reduce the dimension of each document by removing useless elements (characters, images, and advertisements,⁴ etc.).

Therefore, the next necessary step is text cleaning, one of the crucial steps in text analysis. Text cleaning (or text preprocessing) makes an unstructured set of texts uniform across and within and eliminates idiosyncratic characters.⁵ Text cleaning can be loosely divided into a set of steps as shown below.

The text excerpt presented in Section 2 contains some useful information about the content of the discussion, but also a lot of unnecessary details, such as punctuation marks, dates, ubiquitous words. Therefore, the first logical step is to remove punctuation and idiosyncratic characters from the set of texts.

This includes any strings of characters present in the text, such as punctuation marks, percentage or currency signs, or any other characters that are not words. There are two coercing functions⁶ called `content_transformer` and `toSpace` that, in conjunction, get rid of all pre-specified idiosyncratic characters.

⁴Removal of images and advertisements is not covered in this paper.

⁵Specific characters that are not used to understand the meaning of a text.

⁶Many programming languages support the conversion of value into another of a different data type. This kind of type conversions can be implicitly or explicitly made. Coercion relates to the implicit conversion which is automatically done. Casting relates to the explicit conversion performed by code instructions.

The character processing function is called `toSpace`. This function takes a pre-defined punctuation character, and converts it into space, thus erasing it from the text. We use this function inside the `tm_map` wrapper, that takes our corpus, applies the coercing function, and returns our corpus with the already made changes.

In the example below, `toSpace` removes the following punctuation characters: `"-", ",", ".", "`. This list can be expanded and customized (user-defined) as needed.

```
corpus <- tm_map(corpus, toSpace, "-")
corpus <- tm_map(corpus, toSpace, ",")
corpus <- tm_map(corpus, toSpace, ".")
```

The text below shows our original excerpt, with the aforementioned punctuation characters removed:

```
The broad forum discussions took place on December 24 2006 and the narrow
forum discussions on December 25 2006 January 2007 General Before the
Governor makes the monthly interest rate decision discussions are held at
two levels The first discussion takes place in a broad forum in which the
relevant background economic conditions are presented including real and
monetary developments in Israel's economy and developments in the global
economy
```

Another way to eliminate punctuation marks, or characters, is to apply the `removePunctuation` function to the corpus. This function removes a set of pre-defined punctuation characters, but it cannot be customized if the need arises. One can combine both approaches (`toSpace` and `removePunctuation`) in order to effectively remove all punctuation and idiosyncratic characters from text.

Besides, any numbers present in the texts of our corpus can be removed by the `removeNumbers` function, such as in the below code:

```
corpus <- tm_map(corpus, removePunctuation)
corpus <- tm_map(corpus, removeNumbers)
```

Now, the text below shows our original excerpts, but without any punctuation marks or digits:

The broad forum discussions took place on December and the narrow forum discussions on December January General Before the Governor makes the monthly interest rate decision discussions are held at two levels The first discussion takes place in a broad forum in which the relevant background economic conditions are presented including real and monetary developments in Israels economy and developments in the global economy

The current text excerpt conveys the meaning of this meeting a little more clearly, but there is still much unnecessary information. Therefore, the next step would be to remove the so-called stop words from the text.

What are stop words? Words such as “the”, “a”, “and”, “they”, and many others can be defined as stop words. Stop words usually refer to the most common words in a language, and as they are so common, carry no specific informational content. Since these terms do not carry any meaning as standalone terms, they are not valuable for our analysis. In addition to a pre-existing list of stop words, ad hoc stop words can be added to list.

We apply a function from the package `tm` onto our existing corpus as defined above in order to remove the stop words. There is a coercing function called `removeWords` that erases a given set of stop words from the corpus. There are different lists of stop words available, and we use a standard list of English stop words.

However, before removing the stop words, we need to turn all of our existing words within the text into lowercase. Why? Because converting to lowercase, or case folding, allows for case-insensitive comparison. This is the only way for the function `removeWords` to identify the words subject for removal.

Therefore, using the package `tm`, and a coercing function `tolower`, we convert our corpus to lowercase:

```
corpus <- tm_map(corpus, tolower)
```

Below is the example text excerpt following the command mentioned above:

the broad forum discussions took place on december 24 2006 and the narrow forum discussions on december 25 2006 january 2007 general before the governor makes the monthly interest rate decision discussions are held at two levels the first discussion takes place in a broad forum in which the

relevant background economic conditions are presented including real and monetary developments in israel's economy and developments in the global economy

We can now remove the stop words from the text:

```
corpus <- tm_map(corpus, removeWords, stopwords("english"))
```

Here, `tm_map` is a wrapper function that takes the corpus and applies character processing function `removeWords` onto all of the contents of the corpus (all 220 documents). It returns the modified documents in the same format, a corpus, but with the changes already applied. The following output shows our original text excerpt with the stop words removed:

```
broad forum discussions took place december narrow forum discussions  
december january general governor makes monthly interest rate decision  
discussions held two levels first discussion takes place broad forum  
relevant background economic conditions presented including real monetary  
developments israel's economy developments global economy
```

The next and final step is to stem the remaining words. Stemming is a process of turning several words that mean the same thing into one. For example, after stemming, words such as "banking", "banks", and "banked" become "bank". As stemming reduces inflected or derived words to their word stem or root. This can be thought of as word normalization. Stemming algorithm allows us not to count different variations as the same term as separate instances.

Below, we use a coercing function called `stemDocument`, that stems words in a text document using Porter's stemming algorithm. This algorithm removes common morphological and inflectional endings from words in English, as described in the previous paragraph.

```
corpus <- tm_map(corpus, stemDocument)
```

Once we have applied several of these character processing functions to our corpus, we would like to examine it in order to view the results. Overall, as a result of the above procedures, we end up with the following:

```
broad forum discuss took place decemb narrow forum discuss decemb januari
general governor make month interest rate decis discuss held two level
first discuss take place broad forum relev background econom condit present
includ real monetari develop israel economi develop global economi
```

This last text excerpt shows what we end up with once the data cleaning manipulations are done. While the excerpt we end up with resembles its original only remotely, we can still figure out reasonably well the subject of the discussion.⁷

4 Data structures

Once the text cleaning step is done, R allows us to store the results in one of the two following formats, `dtm` and `tidytext`. While there may be more ways to store text, these two formats are the most convenient when working with text data in R. We explain each of these formats next.

4.1 Document Term Matrix

Document Term Matrix (`dtm`) is a mathematical matrix that describes the frequency of terms that occur in a collection of documents. Such matrices are widely used in the field of natural language processing. In `dtm`, each row corresponds to a specific document in the collection and each column corresponds to the specific term within that document. An example of a `dtm` is shown in Table 1.

This type of matrix represents the frequency for each unique term in each document in corpus. In R, our corpus can be mapped into a `dtm` object class by using the function `DocumentTermMatrix` from the `tm` package.

```
dtm <- DocumentTermMatrix(corpus)
```

The goal of mapping the corpus onto a `dtm` is twofold; the first is to present the topic of each document by the frequency of semantically significant and unique terms, and second, to position the corpus for future data analysis.

⁷Another way to perform the changes discussed in this section would be to use the `dplyr` command (see below). Both ways are viable, and we keep the longer explanation within the paper to explain the sequence of steps thoroughly.

```
corpus = corpus %>% tm_map(removePunctuation) %>% tm_map(removeNumbers) %>%
tm_map(tolower) %>% tm_map(removeWords, stopwords("english"))
%>%tm_map(stemDocument)
```


	Term j			
Document i	accord	activ	averag	...
May 2008	3	9	4	...
June 2008	6	4	16	...
July 2008	5	3	7	...
August 2008	4	9	12	...
September 2008	5	8	22	...
October 2008	3	20	16	...
November 2008	6	5	11	...
...

Table 1: An excerpt of a dtm

The value in each cell of this matrix is typically the word frequency of each term in each document. This frequency can be weighted in different ways, to emphasize the importance of certain terms and de-emphasize the importance of others. The default weighting scheme within the `DocumentTermMatrix` function is called Term Frequency (tf). Another common approach to weighting is called Term Frequency - Inverse Document Frequency (tf-idf).

While the tf weighting scheme is defined as the number of times a word appears in the document, tf-idf is defined as the number of times a word appears in the document but is offset by the frequency of the words in the corpus, which helps to adjust for the fact that some words appear more frequently in general.

Why is the frequency of each term in each document important? A simple counting approach such as term frequency may be inappropriate because it can overstate the importance of a small number of very frequent words. Term frequency is the most normalized one, measuring how frequently a term occurs in a document with respect to the document length, such as:

$$\text{tf}(t) = \frac{\text{Number of times term } t \text{ appears in a document}}{\text{Total number of terms in the document}} \quad (1)$$

A more appropriate way to calculate word frequencies is to employ the tf-idf weighting scheme. It is a way to weight the importance of terms in a document based on how frequently they appear across multiple documents. If a term frequently appears in a document, it is important, and it receives a high score. However, if a word appears in many documents, it is not a unique identifier, and it will receive a low score. Eq. 1 shows how words that frequently appear in a single document will be scaled up, and Eq. 2 shows how common words which appear

in many documents will be scaled down.

$$\text{idf}(t) = \ln \left(\frac{\text{Total number of documents}}{\text{Number of documents with term } t \text{ in it}} \right) \quad (2)$$

Conjugating these two properties yield the `tf-idf` weighting scheme.

$$\text{tf-idf}(t) = \text{tf}(t) \times \text{idf}(t) \quad (3)$$

In order to employ this weighting scheme (Eq. 3), we can assign this option within the already familiar function `dtm`:

```
dtm <- DocumentTermMatrix(corpus, control = list(weighting =  
weightTfIdf))
```

The above-mentioned steps provide us with a suitable numeric matrix under the name of `dtm`. In this matrix, each cell contains an integer, corresponding to a (perhaps weighted) number of times a specific term appeared in each document within our corpus. However, most cells in such `dtm` will be empty, i.e., zeros, because most terms do not appear in most of the documents. This property is referred to as matrix sparsity.

Most term-document matrices tend to be high-dimensional and sparse. This is because any given document will contain only a subset of unique terms that appear throughout the corpus. This will result in any corresponding document-row having zeros for terms that were not used in that specific document. Therefore, we need an approach to reduce dimensionality.

Function `RemoveSparseTerms` takes our existing `dtm`, and a certain numerical value called `sparse`, and returns an adjusted `dtm`, where the number of elements is reduced, depending on the value set for `sparse`. This numerical value is defined as the maximum allowed sparsity, as is set in the range of zero to one. In the sense of the `sparse` argument to `RemoveSparseTerms`, sparsity refers to the threshold of relative document frequency of a term, above which the term will be removed. Sparsity becomes smaller as this threshold approaches one.

In a way, this process can be thought of as removing outliers. For example, the code below will yield a `dtm` where all terms appearing in at least 90% of the documents will be kept.

```
dtm <- RemoveSparseTerms(dtm, 0.1)
```

Now, we have a `dtm` that is ready for the initial text analysis. An example of output following this weighting scheme and subsequent sparsity reduction of a certain degree might yield Table 2.

	abroad	acceler	accompani	account	achiev	adjust	...
01-2007	0.0002	0.000416	0.000844	0.000507	0.000271	0.000289	...
01-2008	0.00042	0.000875	0.000887	0.000152	9.49E-05	0.000304	...
01-2009	0.000497	0	0	9.01E-05	0.000112	0.000957	...
01-2010	0.000396	0	0	7.18E-05	8.95E-05	0.000954	...
01-2011	0.000655	0	0.000691	0.000119	7.39E-05	0.000552	...
01-2012	0.000133	0	0.001124	9.65E-05	6.01E-05	0	...
01-2013	0.00019	0.000395	0	0.000138	8.56E-05	0.000274	...
01-2014	0	0.000414	0	0.000144	8.98E-05	0	...
01-2015	0	0.00079	0	6.88E-05	8.57E-05	0.000183	...
01-2016	0	0.000414	0	0	0.00018	0.000192	...
01-2017	0	0.000372	0.000755	6.48E-05	0.000323	0.000689	...
01-2018	0.000581	0	0.002455	0.000211	0	0	...

Table 2: An excerpt of a `dtm` with `tf-idf` weighting methodology. The highest values for the selected sample are highlighted in gray.

4.2 Tidytext Table

`Tidytext` is an R package, detailed in Wickham (2014). This format class was developed specifically for the R software, and for the sole purpose of text mining. `Tidytext` presents a set of documents as a one-term-per-document-per-row data frame first. This is done with the help of the `tidy` function within the `tidytext`.

A `tidytext` structured data set has a specific format: each variable is a column, each observation is a row, and each type of observational unit is a table.

This one-observation-per-row structure is in contrast to the ways text is often stored in current analyses, perhaps as strings or in a `dtm`. For `tidytext`, the observation that is stored in each row is most often a single word, but can also be an n-gram, sentence, or paragraph. There is also a way to convert the `tidytext` format into the `dtm` format. We plan to use the `tidytext` package in one of our extensions to the current project. Instead of analyzing single words within each document, we will conduct our analysis on n-grams, or sets of two, three, or more words, or perhaps sentences.

The `tidytext` package is more limited than `tm`, but in many ways is more intuitive. The `tidytext` format represents a table with one word (or expression) per row. As we will show, this is different from other formats where each word corresponds with the document from which it comes.

For example, Fig. 1 presents the most frequent words in the corpus as produced by the `tidytext` package. The below code takes all of the words that appear in the corpus at least 1200 times or more and plots their frequencies.

```
tidy.table <- tidy.table %>%  
count(word, sort = TRUE) %>%  
filter(n > 1200) %>%  
mutate(word = reorder(word, n)) %>%  
ggplot(aes(word, n)) + geom_col() + xlab(NULL) + coord_flip()
```

In this code, `n` is the word count, i.e., how many times each word appears in the corpus.

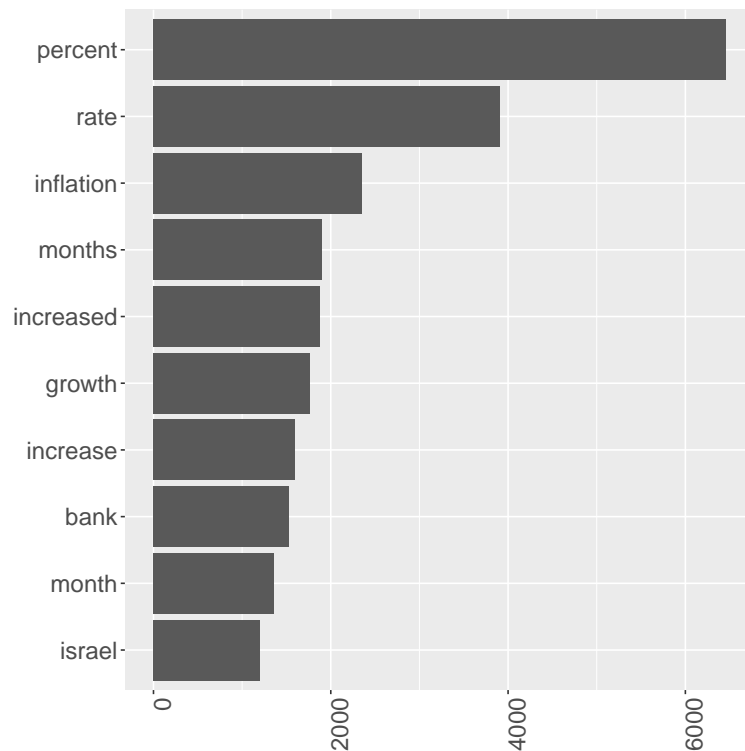


Figure 1: Histogram containing most popular terms within the `tidytext` table.

Besides being more intuitive, the `tidytext` package has the capability for better graphics. An example is provided in Section 4.3.

4.3 Data Exploration

Given a `dtm` with reduced dimensions, as described above, we can apply exploratory analysis techniques to find out about what the corpus, or each document within

the corpus, is talking. As with the text cleaning, there are several logical steps, and the first would be to find out what the most frequent terms within the `dtm` are.

The following piece of code sums up the columns within the `dtm`, and then sorts it in descending order within the data frame called `order.frequencies`. We can then view terms with the highest and the lowest frequencies by using functions `head` and `tail`, respectively:

```
term.frequencies <- colSums(as.matrix(dtm))
order.frequencies <- order(term.frequencies)
```

Table 3 is an example of an output following these commands, showing the top six most frequent words and their corresponding frequencies.

Term <i>j</i>	
percen	9927
rate	9127
increas	5721
month	5132
interest	4861
bank	4039

Table 3: Top Frequent Words in DTM

Another, and perhaps easier way to identify the frequency terms within the `dtm` is to use function `findFreqTerms`, which is a part of the package `tm`. This function returns a list of terms, which meet two ad-hoc criteria of upper and lower bounds of frequency limits:

```
findFreqTerms(dtm, lowfreq = 1800, highfreq = 5000)
```

Here is an example of an output following these commands. Figure 2 shows a histogram of the terms that appear at least 1800 within our corpus.

Another way to look at the most popular terms is to use the `dtm` with the `tf-idf` frequency weighted terms. Figure 3 shows a histogram of some of the most common terms within the corpus, as weighted by the `tf-idf` approach.

Once we know some of the most frequent terms in our corpus, we can explore the corpus further by looking at different associations between some of them.

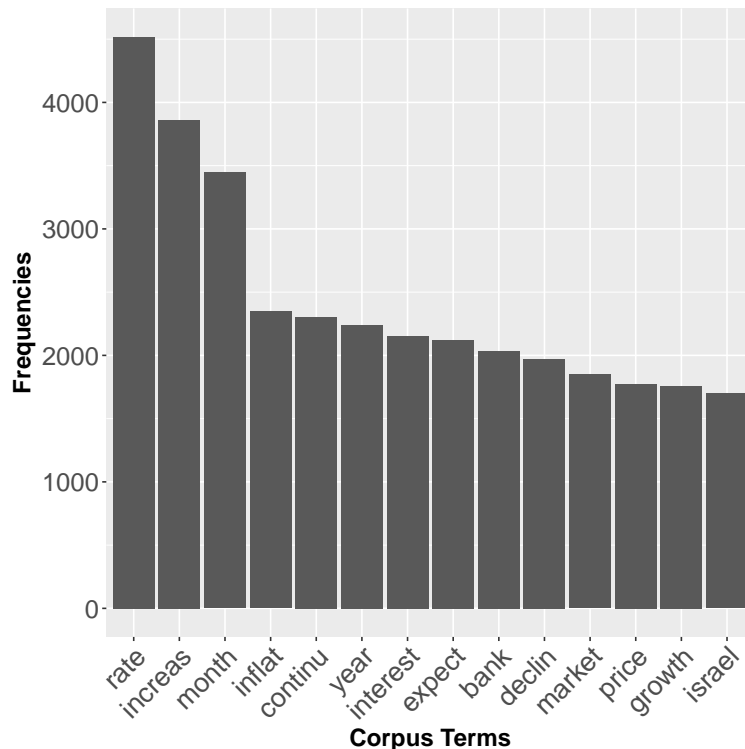


Figure 2: Histogram containing most popular terms within the corpus.

This can be done with the function `findAssocs`, part of the package `tm`. This function takes our `dtm`, a specific term such as “bond”, “increas”, “global”, etc., and an inclusive lower correlation limit as an input, and returns a vector of matching terms and their correlations (satisfying the lower correlation limit `corlimit`):

```
findAssocs(dtm, "bond", corlimit = 0.5)
findAssocs(dtm, "econom", corlimit = 0.35)
findAssocs(dtm, "fed", corlimit = 0.35)
findAssocs(dtm, "feder", corlimit = 0.35)
```

Table 4 is an example of an output following these for the term “bond”:

While this might not be the best way to explore the content of each text or the corpus in general, it might provide some interesting insights for future analysis. The math behind `findAssocs` is based on the standard function `corlimit` in R’s `Stats` package. Given two numeric vectors, `corlimit` computes their correlation.

Another exciting way to explore the contents of our corpus is to create a so-called Wordcloud. A word cloud is an image composed of words used in a particular text or corpus, in which the size of each word indicates its frequency. This can

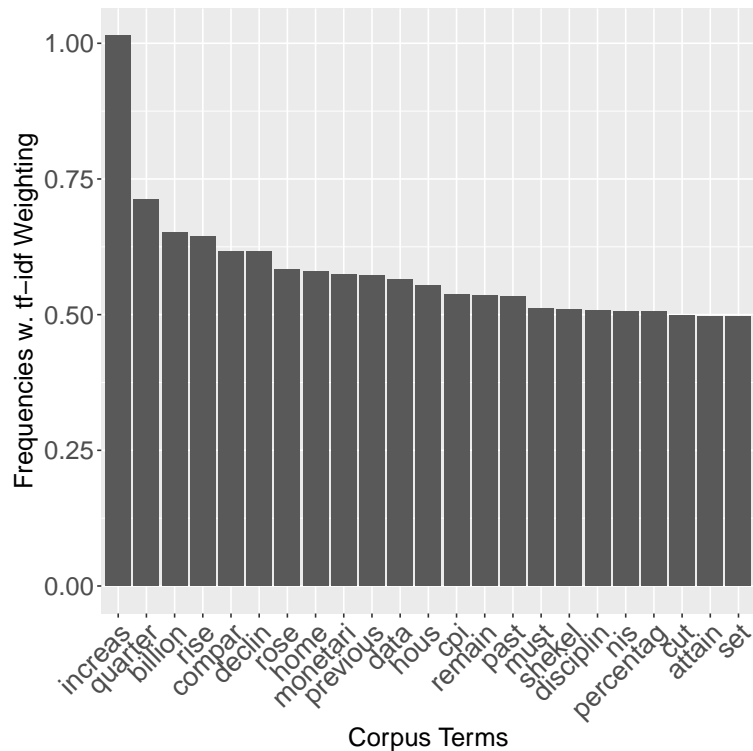


Figure 3: Histogram containing most popular terms within the corpus, with tf-idf weight.

be done with the use of the `wordcloud` package.⁸ Below, we plot word clouds using two different approaches to calculating the frequency of each term in the corpus. The first approach uses a simple frequency calculation.

```
set.seed(142)
wordcloud(names(freq), freq, min.freq = 400, colors=brewer.pal(8,
"Dark2"))
wordcloud(names(freq), freq, min.freq = 700, colors=brewer.pal(8,
"Dark2"))
wordcloud(names(freq), freq, min.freq = 1000, colors=brewer.pal(8,
"Dark2"))
wordcloud(names(freq), freq, min.freq = 2000, colors=brewer.pal(8,
"Dark2"))
```

The function `wordcloud` provides a nice and intuitive visualization of the content of the corpus, or if needed, of each document separately. Fig. 4 to Fig. 6 are

⁸<https://cran.r-project.org/web/packages/wordcloud/wordcloud.pdf>

Term <i>j</i>	
yield	0.76
month	0.62
market	0.61
credit	0.60
aviv	0.58
rate	0.58
bank	0.57
indic	0.57
announc	0.56
sovereign	0.55
forecast	0.55
held	0.54
measur	0.54
treasuri	0.52
germani	0.52
index	0.51

Table 4: Terms most correlated with the term bond

several examples of an output following these commands. For instance, Fig. 4 and Fig. 6 show word clouds containing word terms that appear at least 400 and 2000 times in the corpus, respectively.



Figure 4: Wordcloud containing terms that appear at least 400 times in the corpus.



Figure 5: Wordcloud containing terms that appear at least 700 times in the corpus.

Another way to demonstrate the frequency of terms within the corpus is to use the $tf-idf$ weighting scheme, and produce similar figures. It is clear that with the new weighting scheme, other terms are emphasized more. As an example, Fig. 7 and Fig. 8 show word clouds with word frequencies above 0.06, 0.08 and 0.1.

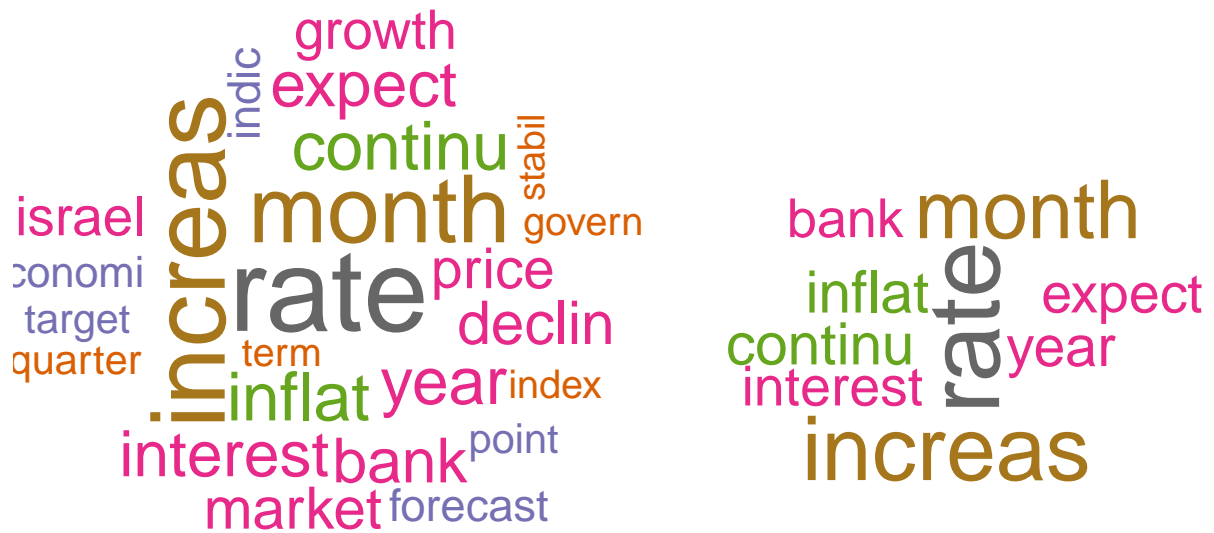


Figure 6: Wordcloud containing terms that appear at least 1000 and 2000 times in the corpus (left and right panel, respectively).



Figure 7: Word cloud containing word terms with word frequencies above 0.06.



Figure 8: Word cloud containing word terms with word frequencies above 0.08 (top panel) and 0.1 (bottom panel).

Another way to explore the corpus content is to apply a clustering algorithm to visualize it with a type of dendrogram or adjacency diagram. The clustering method can be thought of as an automatic text categorization. The basic idea behind document or text clustering is to categorize documents into groups based on likeness. One of the possible algorithms would be to calculate the Euclidean, or geometric, distance between the terms. Terms are then grouped on some distance related criteria.

One of the most intuitive things we can build is correlation maps. Correlation maps show how some of the most frequent terms relate to each other in the corpus, based on some ad-hoc correlation criteria. Below is the code example that can create a correlation map for a given dtm. To plot this object, one will need to use the Rgraphviz package.⁹

```
correlation.limit <- 0.6
frequency.terms.dtm <- findFreqTerms(dtm.tf.idf)
plot(dtm.tf.idf, term = frequency.terms.dtm,
corThreshold=correlation.limit)
```

We plot Fig. 9 and Fig. 10 following the above code.

Another way to visualize relationships between terms within the corpus is to create a dendrogram for each of our dtm using the following commands:

```
dendogram <- dist(t(dtm.sparse.01), method = "euclidian")
dendogram.fit <- hclust(d = dendogram, method = "ward.D")
plot(dendogram.fit, hang = -1)
```

This cluster analysis map is based on the Ward's method, which is based on the original function approach. It relies on choosing a minimum variance criterion that would minimize the total within-cluster variance.¹⁰

The following dendograms, Fig. 11 and Fig. 12, are built based on these relationships:

⁹Available at: <http://bioconductor.org/biocLite.R> or through R console such as:

```
install.packages("BiocManager")
BiocManager::install("Rgraphviz")
```

¹⁰See: Ward, J. H., Jr. (1963), "Hierarchical Grouping to Optimize an Objective Function", Journal of the American Statistical Association, 58, 236–244.



Figure 9: Correlation map using dtm with simple counting weighting scheme.

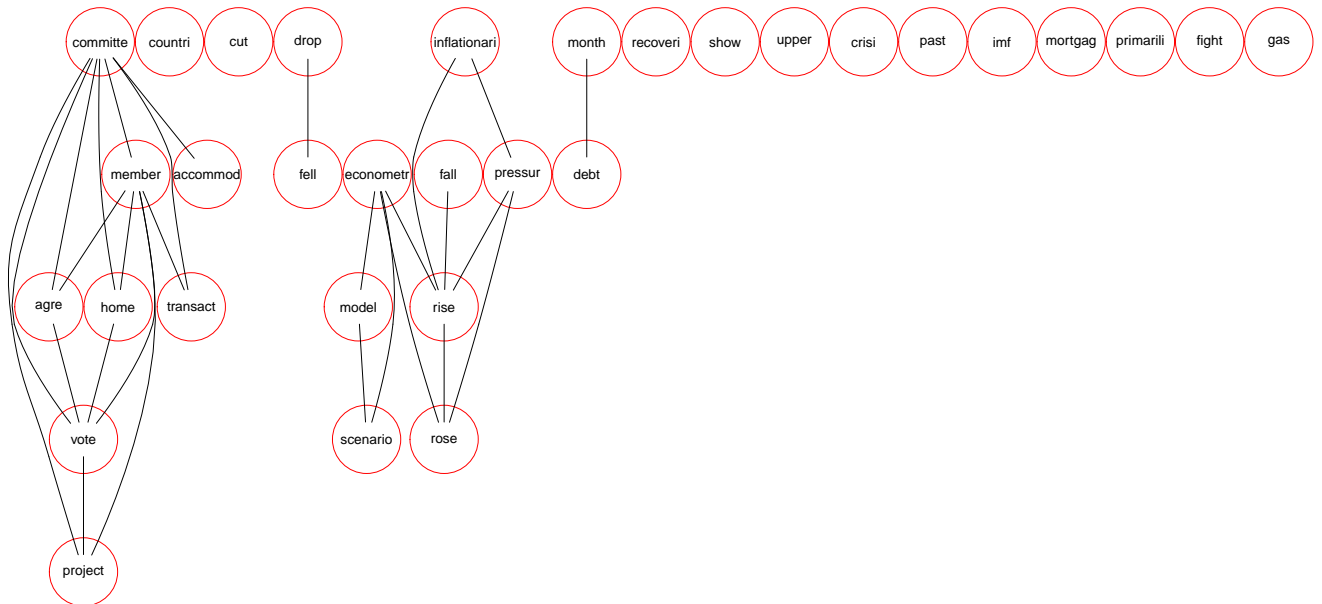


Figure 10: Correlation map using dtm w. tf-idf frequency.

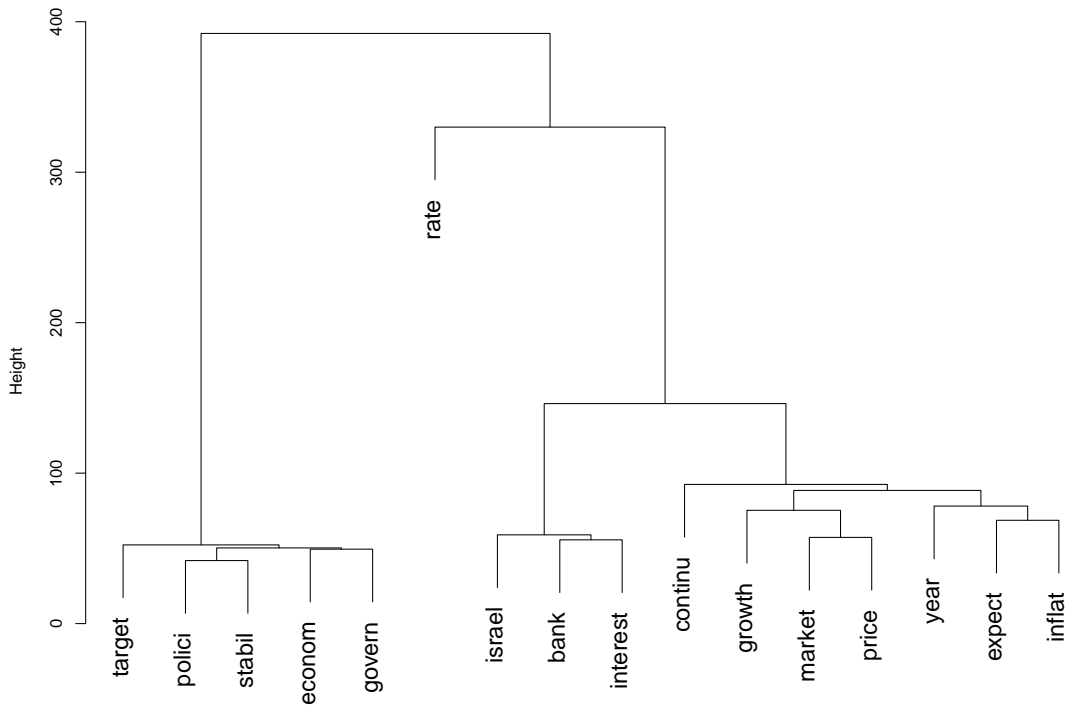


Figure 11: Dendrogram w. frequency.

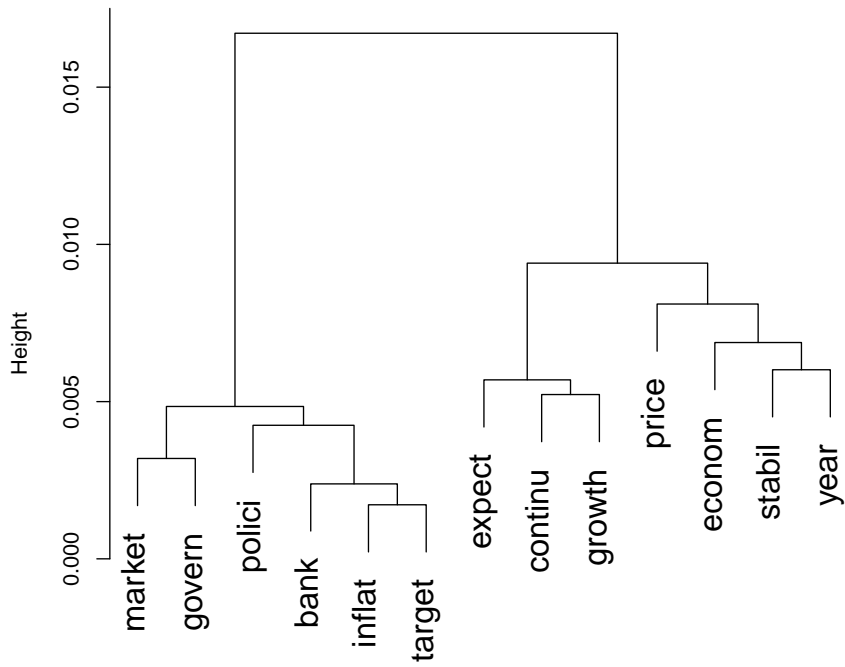


Figure 12: Dendrogram w. tf-idf frequency.

Figure 12 shows a dendrogram based on the dtm weighted using tf-idf scheme.

Another quick and useful visualization of each document's content is allowed by heat maps. Heat maps can be used to compare the content of each document, side by side, with other documents in the corpus, revealing interesting patterns and time trends.

Fig. 13 presents word frequencies for the word list on the bottom of the heatmap. It demonstrates a simple distribution of word frequencies throughout time. For example, the term *accommod* was used heavily during the discussions that took place in mid and late 2001; however, it was not mentioned at all in early 2002.

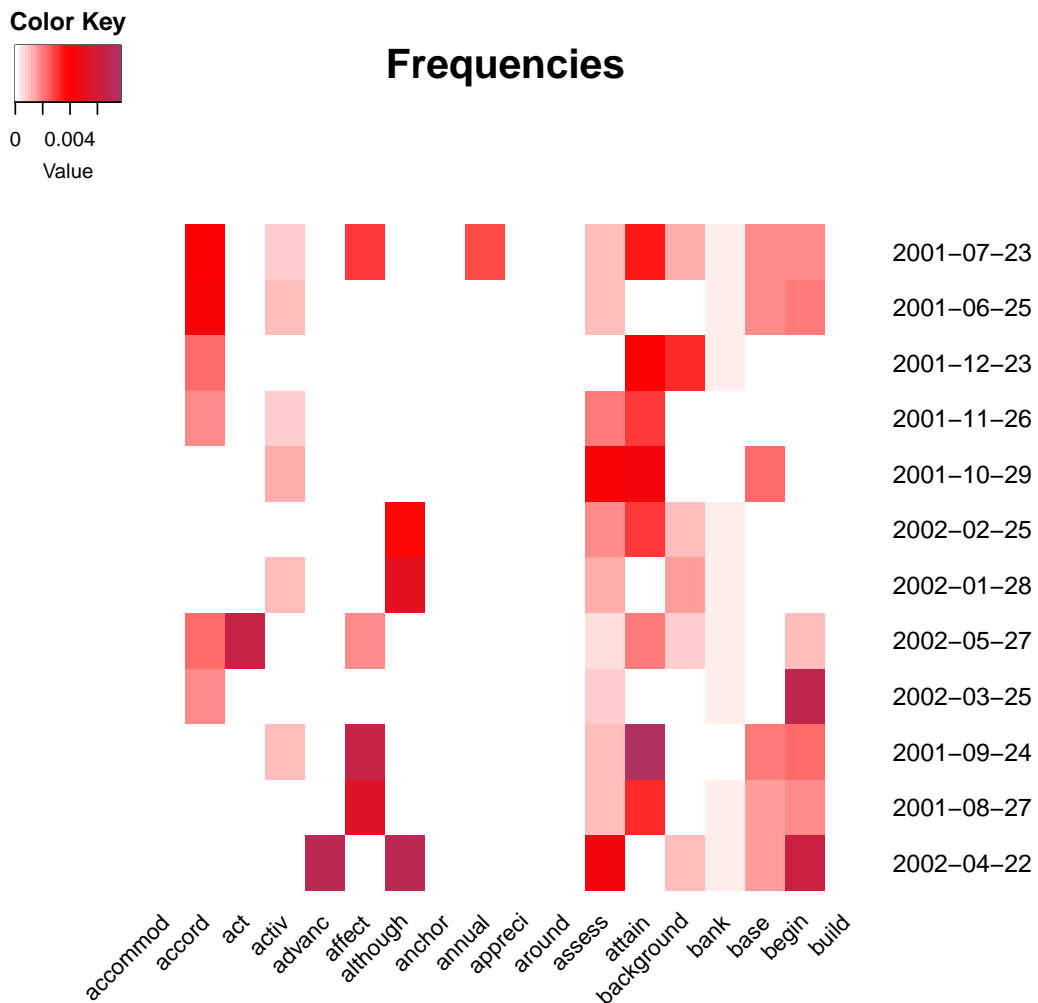


Figure 13: Heatmap for documents published in 2010 (tf-idf weighted dtm). The color key corresponds to probabilities for each topic being discussed during the corresponding interest rate decision meeting.

Fig. 14 presents a heatmap of word frequencies for the period spanning the mid-1999 to early 2000. For example, the term `inflat`, representing discussion around inflation, shows that this topic was discussed heavily in early 2000, in particular in January of 2000. These kinds of figures provide a quick and visual representation of any given interest rate discussion.

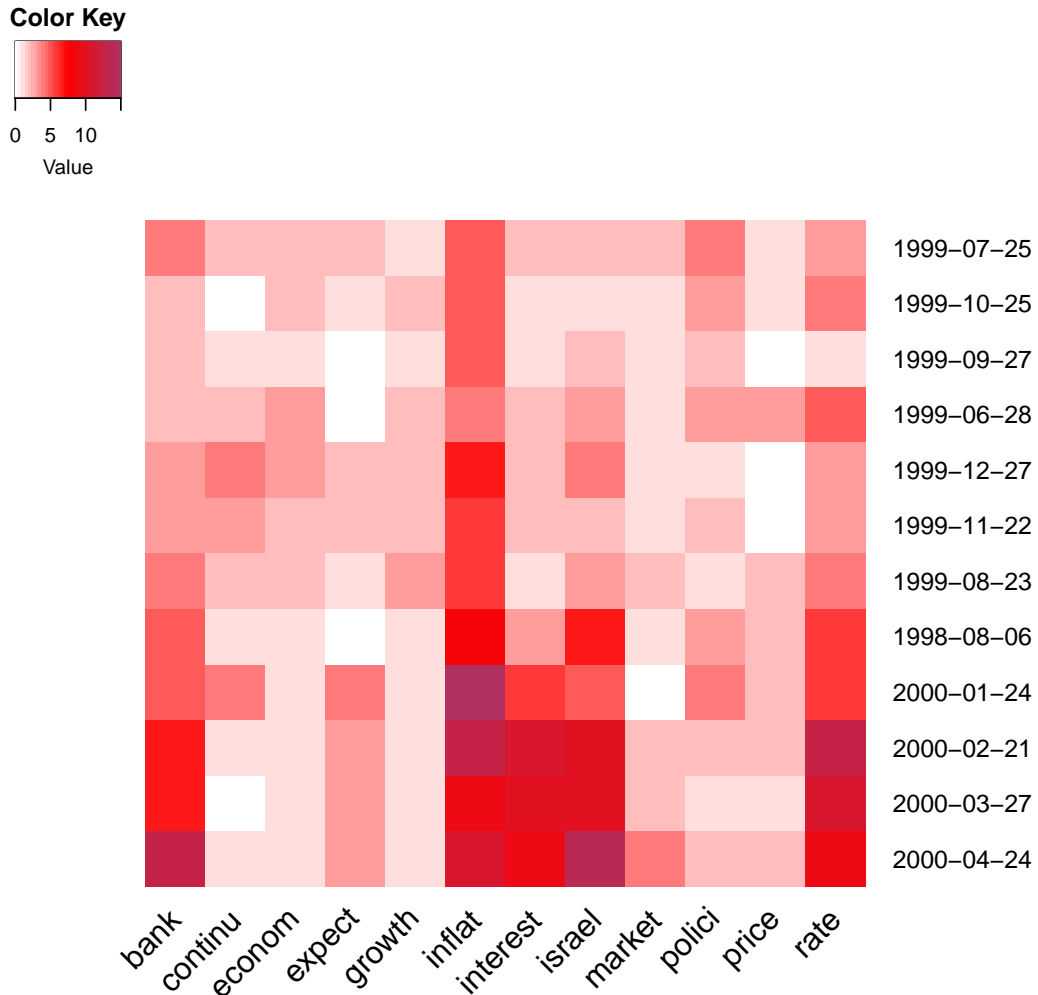


Figure 14: Heatmap for documents published in 1999 (τf weighted dtm). The color key corresponds to probabilities for each topic being discussed during the corresponding interest rate decision meeting.

This section sums up some of the most popular techniques for exploratory text analysis. We show different ways a set of texts can be summarized and visualized easily and intuitively.

5 Text Analytics

The subsequent steps of our analysis can be roughly divided by purpose, analysis within texts and analysis between texts. Techniques such as dictionary and applying various weighting schemes to existing terms can be used for the first purpose. The second group is used for comparison between texts and refers to techniques related to Latent Semantic Analysis (LSA) and Latent Dirichlet Allocation (LDA). We use a specific dictionary methodology for the first goal and a wordscores algorithm, which is an LDA methodology, for the second goal. We describe these techniques in more detail in this section. The majority of text analytic algorithms in R are written with the `dtm` format in mind. For this reason, we will use `dtm` format in order to discuss the application of these algorithms.

5.1 Word Counting

Dictionary-based text analysis is a popular technique due to its simplicity. Dictionary-based text analysis begins with setting a predefined list of words that are relevant for the analysis of that particular text. For example, the most commonly used source for word classifications in the literature is the Harvard Psycho-sociological Dictionary, specifically, the Harvard-IV-4 TagNeg (H4N) file.

However, word categorization for one discipline (for example, psychology) might not translate effectively into another discipline (for example, economics or finance). Therefore, one of the drawbacks of this approach is the importance of adequately choosing an appropriate dictionary or a set of predefined words. Loughran and McDonald (2011) demonstrate that some words that may have a negative connotation in one context may be neutral in others. The authors show that dictionaries containing words like tax, cost, or liability that convey negative sentiment in a general context, are more neutral in tone in the context of financial markets. The authors construct an alternative, finance-specific dictionary to reflect tone in a financial text better. They show that, with the use of a finance-specific dictionary, they can predict asset returns better than other, generic, dictionaries. We use the Loughran and McDonald (2011) master dictionary, which is available on their website. We divide the dictionary into two separate csv files into two sentiment categories. Each file contains one column with several thousand words; one is a list of positive terms, and one is a list of negative terms. We read in both of these files into R as csv files.

```
dictionary.finance.negative <- read.csv("negative.csv",
```

```
stringsAsFactors = FALSE)[,1]
dictionary.finance.positive <- read.csv("positive.csv",
stringsAsFactors = FALSE)[,1]
```

For example, a word is a positive term if it belongs to a positive, or "hawkish" category: for example, "increas", "rais", "tight", "pressur", "strength", etc. A word is a negative term if it belongs to a negative or "dovish" category: for example, "decreas", "lower", "loos", "unsatisfi", "worse", etc. A document is classified as positive if the count of positive words is greater than or equal to the count of negative words. Similarly, a document is classified as negative if the count of negative words is greater than the count of positive words. The constructed indicator is presented in Fig. 16. We transform both files into two separate data frames, using function `data.frame`. This function is used for storing data tables in a matrix form. We apply the same text cleaning manipulation to the dictionary words as applied to the corpus texts themselves. The code below applies the text data cleaning principles to the two sentiment dictionaries that we have uploaded. The cleaning involves turning all terms within both dictionaries to lowercase, stemming all of the terms, and dropping all duplicate terms:

```
dictionary.negative <- tolower(dictionary.negative)
dictionary.negative <- stemDocument(dictionary.negative)
dictionary.negative <- unique(dictionary.negative)
```

We then use the `match` function that compares the terms in both dictionaries with each term in the corpus. This function returns a value of one if there is a match, and a value of zero if there is no match. This allows us to calculate the number of times each positive and each negative term appeared in the corpus. We proceed to calculate the relative frequency of each dictionary terms. The code below captures the list of terms from the `dtm` by using the function `colnames` and then matches each of the terms in the corpus with the terms in each of the dictionaries, calculating the total amount of matches for each dictionary:

```
corpus.terms = colnames(dtm)
positive.matches = match(corpus.terms, dictionary.positive, nomatch=0)
negative.matches = match(corpus.terms, dictionary.negative, nomatch=0)
```

We then assign a value of one to each positive term (P) in the document, and a value of minus one to each negative term (N) in a document, and measure the overall sentiment score for each document i by the following formula:

$$Score_i = \frac{P_i - N_i}{P_i + N_i} \in [-1; 1] \quad (4)$$

A document is classified as positive if the count of positive words is greater than or equal to the count of negative words. Similarly, a document is negative if the count of negative words is greater than the count of positive words. The code below demonstrates a simple calculation of this indicator:

```
document.score = sum(positive.matches) - sum(negative.matches)
scores.data.frame = data.frame(scores = document.score)
```

Fig. 15 presents the main indicators constructed using the dictionary word count.

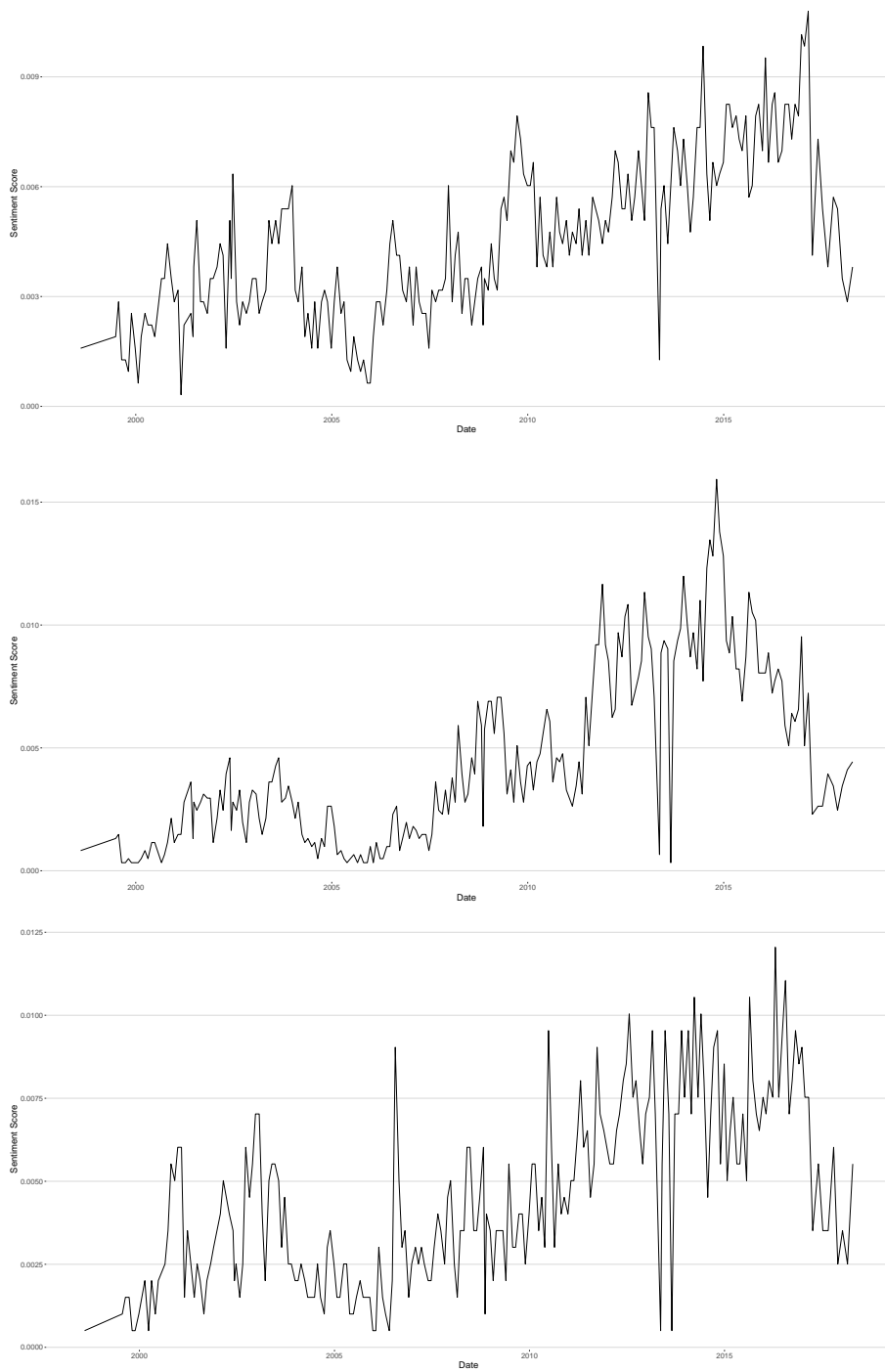


Figure 15: Scaled count of positive (top panel), negative (middle panel), and uncertainty (bottom panel) words in each document using the dictionary approach.

Using the positive and negative sentiment indicators exposed in Fig. 15, Fig. 16 shows the simple dictionary based sentiment indicator.

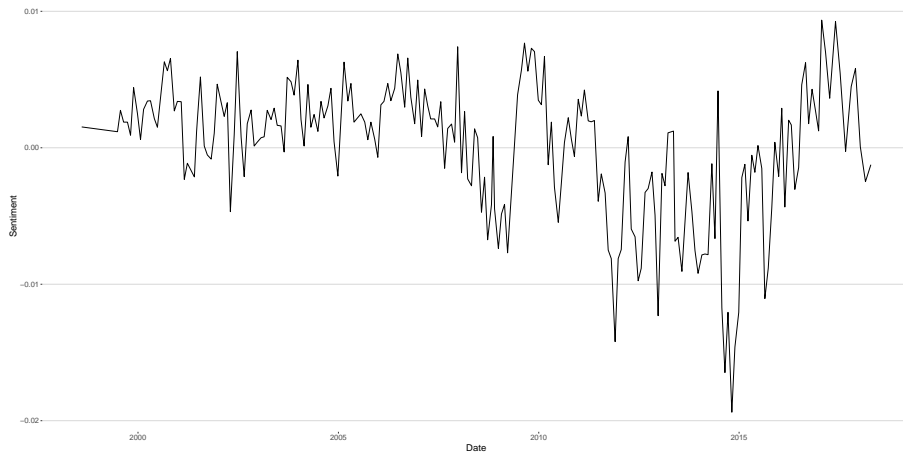


Figure 16: Sentiment indicator built using the dictionary approach.

Fig. 17 demonstrates a distribution of positive and negative matches throughout the corpus, as produced by the package tidytext.

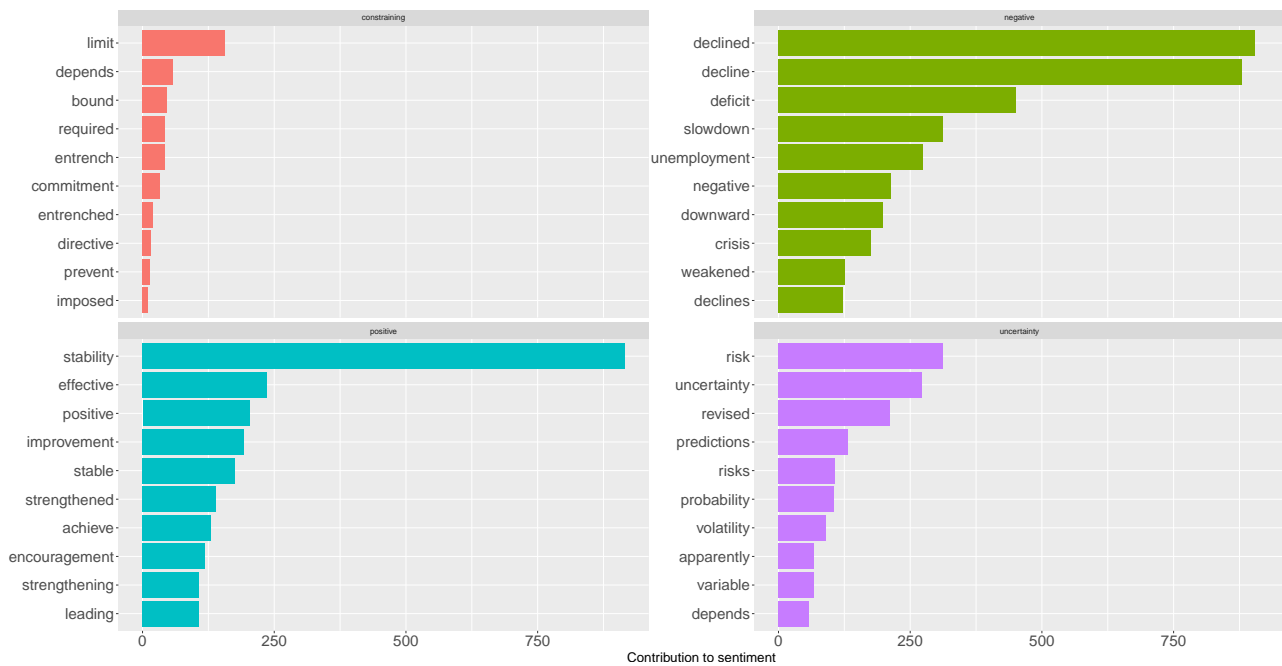


Figure 17: How much each term contributes to the sentiment in each corresponding category. These categories are defined as mutually exclusive. Constraining (top left), positive (top right), negative (bottom left), and uncertainty (bottom right) sentiments are represented.

To sum it up, this is a “quick and dirty” way to summarize the sentiment of any given document. The strength of this approach is that it is intuitive, and easy to implement. In addition, any given dictionary that is being used for document scoring can be customized with *ad-hoc* words, related to the subject matter. This, however, opens the door to a potential weakness of this approach. There is a point where a customized dictionary list might lose its objectivity. Dictionary-based sentiment measurement is the first step in the sentiment extraction process.

5.2 Relative Frequency

An algorithm called *wordscores* estimates policy positions by comparing sets of texts using the underlying relative frequency of words. This approach, described by Laver et al. (2003), proposes an alternative way to locate the policy positions of political actors by analyzing the texts they generate. Mainly used in political sciences, it is a statistical technique for estimating the policy position based on word frequencies. The underlying idea is that relative word usage within documents should reveal information of policy positions.

The algorithm assigns policy positions (or “scores”) to documents on the basis of word counts and known document scores (reference texts) via the computation of “word scores”. One assumption is that their corpus can be divided into two sets (Laver et al., 2003). The first set of documents has a political position that can be either estimated with confidence from independent sources or assumed uncontroversial. This set of documents is referred to as the “reference” texts. The second set of documents consists of texts with unknown policy positions. These are referred to as the “virgin” texts. The only thing known about the virgin texts is the words in them, which are then compared to the words observed in reference texts with known policy positions.

One example of a reference text describes the interest rate discussion meeting that took place on November 11, 2008. We chose this text as a reference because it is a classic representation of dovish rhetoric. The excerpt below mentions a negative economic outlook, both in Israel and globally, and talks about the impact of this global slowdown on real activity in Israel:

Recently assessments have firmed that the reduction in global growth will be more severe than originally expected. Thus, the IMF significantly reduced its growth forecasts for 2009: it cut its global growth forecast by 0.8 percentage points to 2.2 percent, and its forecast of the increase in world trade by 2 percentage points, to

2.1 percent. These updates are in line with downward revisions by other official and private-sector entities. The increased severity of the global slowdown is expected to influence real activity in Israel. The process of cuts in interest rates by central banks has intensified since the previous interest rate decision on 27 October 2008.

Another example of the reference text describes the interest rate discussion meeting that took place on June 24, 2002. This text is a classic representation of hawkish rhetorics. For example, the excerpt below mentions a sharp increase in inflation and inflation expectations:

The interest-rate hike was made necessary because, due to the rise in actual inflation since the beginning of the year and the depreciation of the NIS, inflation expectations for the next few years as derived from the capital market, private forecasters, and the Bank of Israel's models have also risen beyond the 3 percent rate which constitutes the upper limit of the range defined as price stability. Despite the two increases in the Bank of Israel's interest rate in June, inflation expectations for one year ahead have risen recently and reached 5 percent.

Specifically, the authors use relative frequencies observed for each of the different words in each of the reference texts to calculate the probability that we are reading a particular reference text, given that we are reading a particular word. This makes it possible to generate a score of the expected policy position of any text, given only the single word in question.

Scoring words in this way replaces and improves upon the predefined dictionary approach. It gives words policy scores, without having to determine or consider their meanings in advance. Instead, policy positions can be estimated by treating words as data associated with a set of reference texts.¹¹

In our analysis, out of the sample containing 224 interest rate statements, we pick two reference texts that have a pronounced negative (or "dovish") position and two reference texts that have a pronounced positive (or "hawkish") position regarding the state of the economy during the corresponding month. We assign

¹¹However, one must consider the possibility that there would be a change in rhetoric over time. Perhaps it would make sense to re-examine the approach at certain points in time. This would depend on the time span of the data.

the score of minus one to the two “dovish” reference texts and the score of one to the two “hawkish” reference texts. We use these known scores to infer the score of the virgin, or out of sample, texts. Terms contained by the out of sample texts are compared with the words observed in reference texts, and then each out of sample text is assigned a score, $Wordscore_i$.

In R, we utilize the package `quanteda`, which contains the function `wordfish`. This function takes a predefined corpus and applies the `wordscores` algorithm as described above. Once the selection process of the reference documents is complete, the code is fairly simple.

```
wordscore.estimation.results <- wordfish(corpus, dir = c(1,5))
```

This function takes our corpus as an input, as well as the two selected reference documents (here, document number one and document number five), and returns a set of estimation position, as related to each document.

5.3 Topic Models

Topic modeling describes a method for classifying and then mapping a collection of documents into a given number of topics, that best represents the information given. Largely, topic modeling is a set of algorithms that automatically find and classify recurring patterns of words throughout a set of documents. In this paper, we use the LDA algorithm for the above task, which is a widely used algorithm in the field of computer science (Blei et al., 2003).

This algorithm assumes that each of the documents within our corpus consists of a mixture of *corpus-wide* topics. These topics, however, are not observable but are instead hidden behind everyday words and sentences. Specifically, LDA estimates what fraction of each document in a corpus can be assigned to each of the topics. The number of topics is set in advance. We do not observe the topics in the document, only the words that those topics tend to generate. LDA is an algorithm that employs an unsupervised learning approach, in that we do not set prior probabilities for any of the words belonging to any given topic. Besides, it is a mixed-membership model, and therefore the assumption is that every word in the corpus simultaneously belongs to several topics and the topic distributions vary over documents in the corpus.

To provide more intuition, consider an implicit assumption that a given set of words relates to a specific topic. For example, consider the following set of words:

gain, employment, labor. Each of these words would map into an underlying topic “labor market” with a higher probability compared to what it would map into the topic of “economic growth”. This algorithm has a considerable advantage, its objectivity. It makes it possible to find the best association between words and the underlying topics without preset word lists or labels. The LDA algorithm works its way up through the corpus. It first associates each word in the vocabulary to any given latent topic. It allows each word to have associations with multiple topics. Given these associations, it then proceeds to associate each document with topics. Besides the actual corpus, the main input that the model receives is how many topics there should be. Given those, the model will generate β_k topic distributions, the distribution over words for each topic. The model will also generate θ_d document distributions for each topic, where d is the number of documents. This modeling is done with the use of Gibbs sampling iterations, going over each term in each document and assigning relative importance to each instance of the term.

In R, we use the package `topicmodels`, with the default parameter values supplied by the LDA function. Specifying a parameter is required before running the algorithm, which increases the subjectivity level. This parameter, k , is the number of topics that the algorithm should use to classify a given set of documents. There are analytical approaches to decide on the values of k , but most of the literature set it on an *ad hoc* basis. When choosing k we have two goals that are in direct conflict with each other. We want to correctly predict the text and be as specific as possible to determine the number of topics. Yet, at the same time, we want to be able to interpret our results, and when we get too specific, the general meaning of each topic will be lost. Hence, the trade-off.

Let us demonstrate with this example by first setting $k = 2$, meaning that we assume only two topics to be present throughout our interest rate discussions. Below are the top seven words to be associated with these two topics.

It can be seen below that while these two sets of words differ, they both have overlapping terms. This demonstrates the idea that each word can be assigned to multiple topics, but with a different probability.

Table 5 shows that Topic 1 relates directly and clearly to changes in the target rate, while Topic 2 relates more to inflationary expectations. However, these are not the only two things that the policymakers discuss during interest rate meetings, and we can safely assume that there should be more topics considered, meaning k should be larger than two.¹²

To demonstrate the opposite side of the trade-off, let us consider $k = 6$, i.e., we

¹²The supervised approach may help to determine the main theme of each topic objectively.

Topic 1	Topic 2
"increas"	"rate"
"rate"	"interest"
"month"	"expect"
"continu"	"israel"
"declin"	"inflat"
"discuss"	"bank"
"market"	"month"
"monetari"	"quarter"

Table 5: Words with the highest probability of appearing in Topic 1 and Topic 2.

assume six different topics are being discussed. Below is the top seven words with the highest probability to be associated with these six topics:

Topic 1	Topic 2	Topic 3	Topic 4	Topic 5	Topic 6
"declin"	"bank"	"increas"	"continu"	"quarter"	"interest"
"monetari"	"economi"	"month"	"rate"	"year"	"rate"
"discuss"	"month"	"interest"	"remain"	"rate"	"israel"
"rate"	"forecast"	"inflat"	"market"	"growth"	"inflat"
"data"	"market"	"hous"	"term"	"month"	"expect"
"polic"	"govern"	"continu"	"year"	"expect"	"discuss"
"indic"	"global"	"rate"	"price"	"first"	"bank"
"develop"	"activ"	"indic"	"growth"	"point"	"econom"

Table 6: Words with the highest probability of appearing in Topics 1 through 6.

The division between topics is less clear in Table 6 compared to Table 5. While Topics 1, 2 and 3 relate to potential changes in interest rate, Topic 4 relates to housing market conditions, and Topic 5 relates to a higher level of expected growth taking into account monetary policy considerations. Topic 6 covers economic growth and banking discussions.

We see that while we get more granularity in topics by increasing the possible number of topics, we see increased redundancy in the number of topics. Given this outcome, we could continue to adjust k and assess the result.

We demonstrate how we run this algorithm. First, we specify a set of parameters for Gibbs sampling. These include `burnin`, `iter`, `thin`, which are the parameters related to the amount of Gibbs sampling draws, and the way these are drawn.

```
burnin <- 4000
iter <- 2000
```

```
thin <- 500
seed <- list(2003, 5, 63, 100001, 765)
nstart <- 5
best <- TRUE
```

As discussed, the number of topics is decided arbitrarily, as an educated guess, and can be adjusted as needed. Here, based on the previous analysis we take the average of the two previous assumptions and end up with four assumed topics.

```
k <- 4
```

The code below runs the LDA algorithm, using the set of parameters as described above and our dtm:

```
lda.results <- LDA(dtm.sparse, k, method = "Gibbs", control =
list(nstart = nstart, seed = seed, best = best, burnin = burnin, iter
= iter, thin = thin))
```

These last lines write out and save the estimated topics as provided by the LDA algorithm:

```
lda.topics <- as.matrix(topics(lda.results))
lda.results.terms <- as.matrix(terms(lda.results,8))
lda.results.terms
```

Let us examine the topics presented in Table 7. Topic 1 relates to current changes in interest rate and its goal of keeping inflation in range. It mentions the interest rate, inflation expectations and the range of inflation. Topic 2 relates to the actual inflation data and inflationary expectations. Topic 3 relates to a high-level monetary policy discussion, and Topic 4 relates to housing market conditions.

LDA algorithm generates probability distribution of topics over the corpus.

Fig. 18 is a heat map containing a sample for the period of June 2007 to April 2008. Given an assumption that only four topics are discussed during each interest rate meeting, the values presented in the legend are probabilities for each topic being discussed during the corresponding interest rate decision meeting.

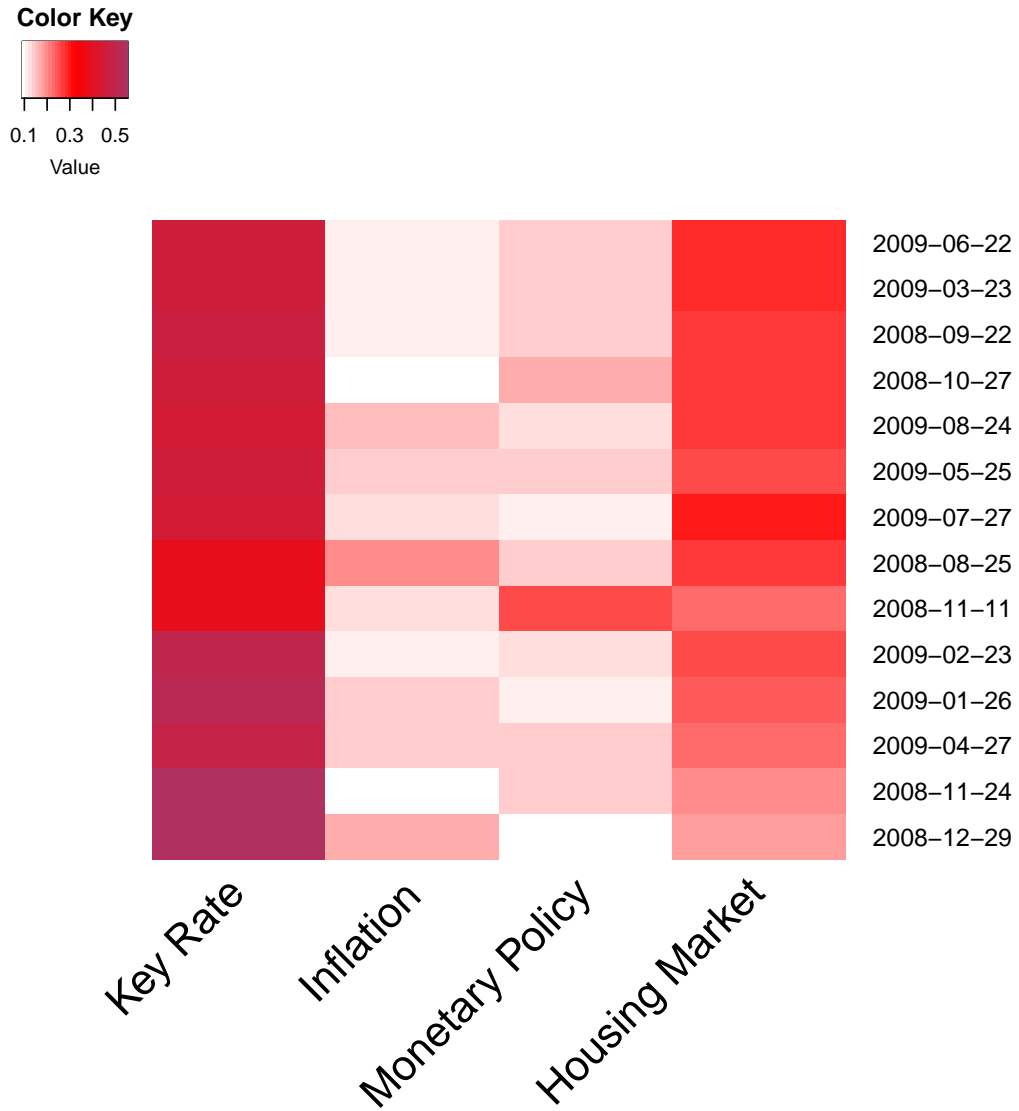


Figure 18: Probability distribution of Topics 1 through 4 over a set of documents from 2007 and 2008. The color key corresponds to probabilities for each topic being discussed during the corresponding interest rate decision meeting.

Topic 1	Topic 2	Topic 3	Topic 4
"expect"	"increas"	"interest"	"month"
"continu"	"declin"	"rate"	"increas"
"rate"	"continu"	"stabil"	"rate"
"inflat"	"rate"	"israel"	"forecast"
"interest"	"expect"	"bank"	"bank"
"rang"	"remain"	"inflat"	"indic"
"israel"	"growth"	"market"	"growth"
"last"	"term"	"govern"	"year"
"price"	"nis"	"year"	"previous"
"bank"	"year"	"target"	"index"
"econom"	"data"	"term"	"hous"

Table 7: Words with the highest probability of appearing in Topics 1 through 4.

For example, during the meeting of November 2008, the "Monetary Policy" topic was discussed with greater probability compared to the "Inflation" topic. As can be seen from Fig. 18, this occurrence stands out from the regular pattern.

Fig. 19 and Fig. 20 present the heat maps about the interest rate announcements during the year 2007 and 2000, respectively.

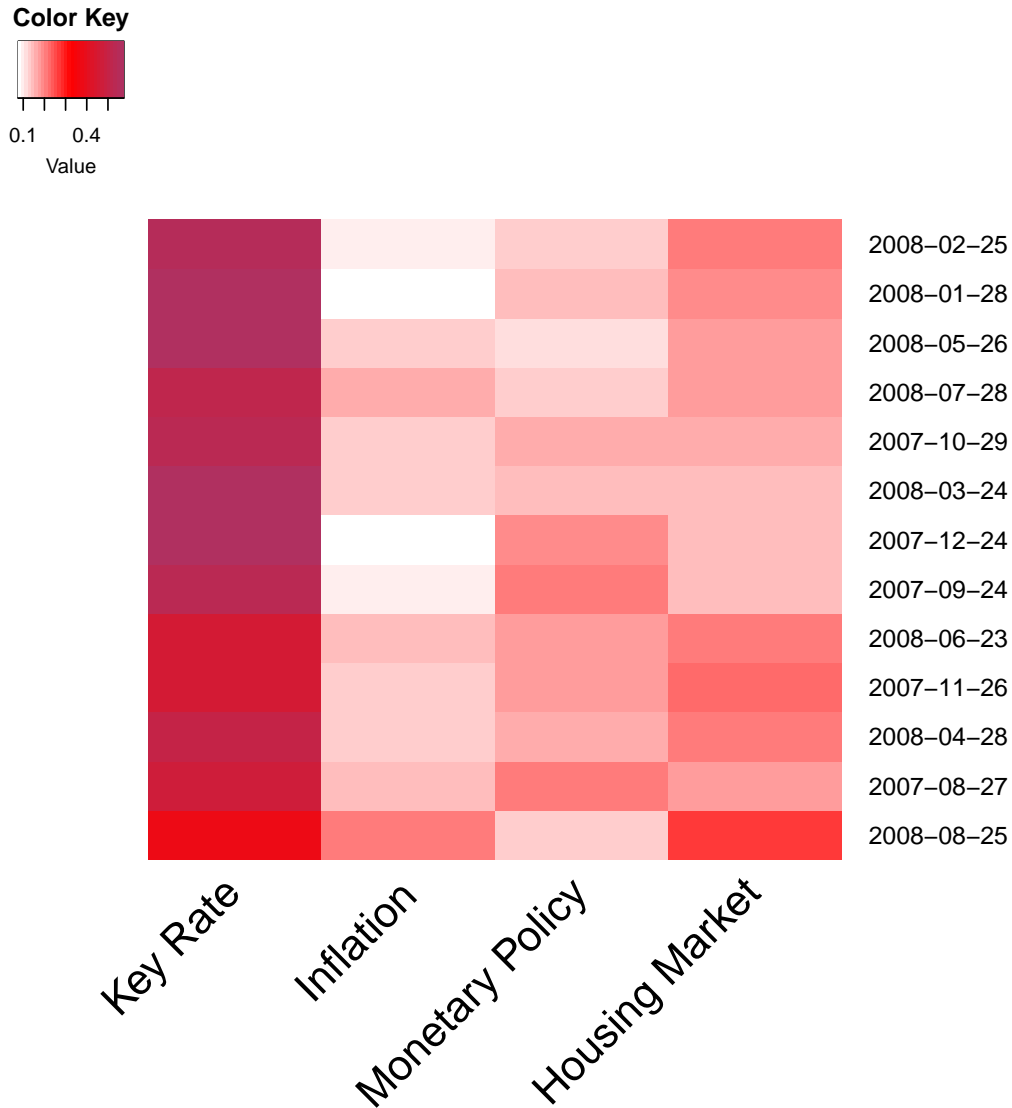


Figure 19: Probability distribution of Topics 1 through 4 over a set of documents from 2008 and 2009. The color key corresponds to probabilities for each topic being discussed during the corresponding interest rate decision meeting.

Figure Fig. 19 shows that in a given set of documents, the bulk of the discussion was spent on discussing the key interest rate set by the Bank of Israel. In contrast, it can be seen that inflation was not discussed at all during certain periods.

Figure Fig. 20 shows that the subject of discussion was mainly monetary policy during this period of time.

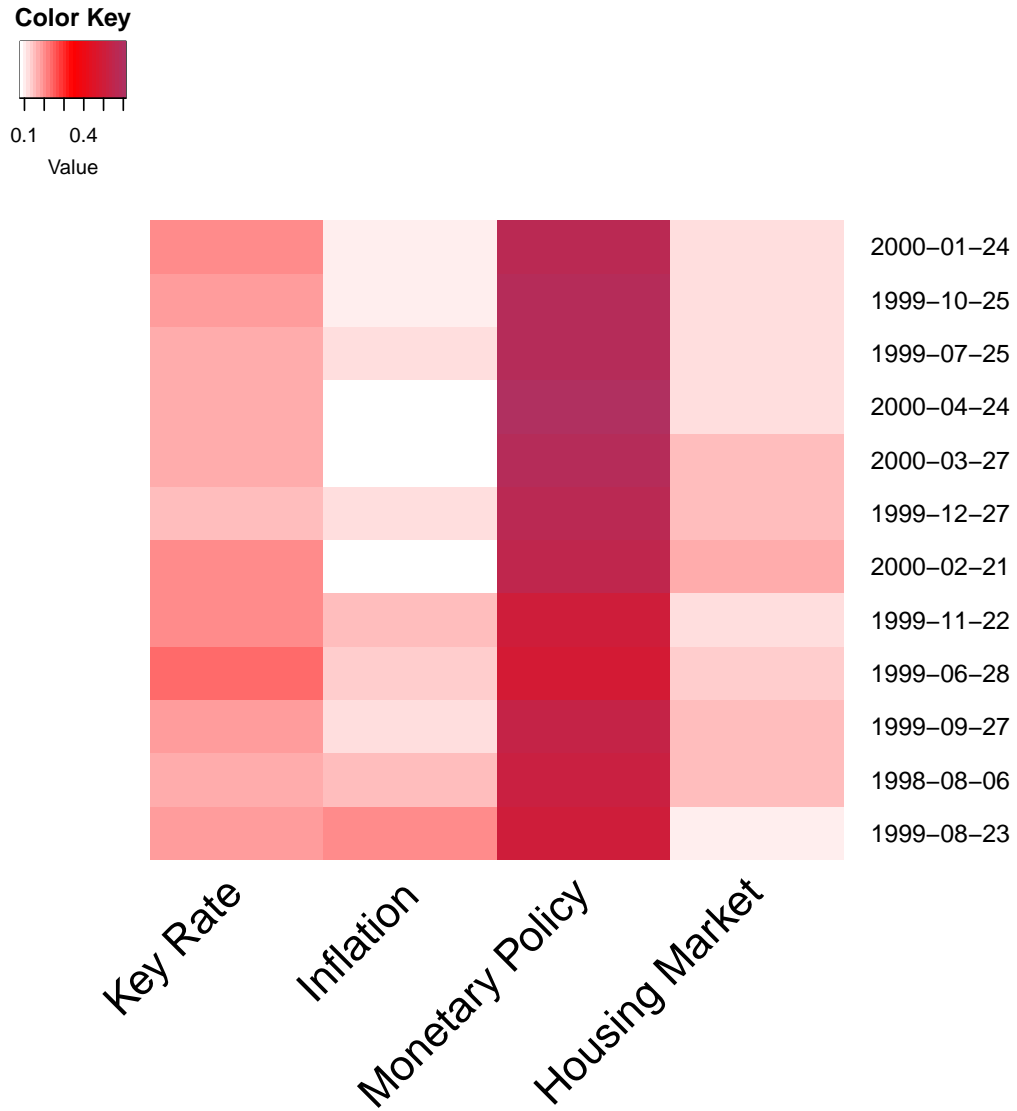


Figure 20: Probability distribution of Topics 1 through 4 over a set of documents from 1999 and 2000. The color key corresponds to probabilities for each topic being discussed during the corresponding interest rate decision meeting.

6 Conclusion

In this paper, we review some of the primary text mining methodologies. We demonstrate how sentiments and text topics can be extracted from a set of text sources. Taking advantage of the open source software package R, we provide a detailed step by step tutorial, including code excerpts that are easy to implement, and examples of output. The framework we demonstrate in this paper shows how to process and utilize text data in an objective and automated way.

As described, the ultimate goal of text analysis is to uncover the information hidden in monetary policymaking and its communication and to be able to organize it consistently. We first show how to set up a directory and input a set of relevant files into R. We show how to store this set of files as a *corpus*, an internal R framework that allows for easy text manipulations. We then describe a series of text cleaning manipulations that sets the stage for further text analysis. In the second part of the paper, we demonstrate approaches to preliminary text analysis and show how to create several summary statistics for our existing corpus. We then proceed to describe two different approaches to text sentiment extraction, and one approach to topic modeling.

We also consider term-weighting and contiguous sequence of words (n-grams) to capture the subtlety of central bank communication better. We consider field-specific weighted lexicon, consisting of two, three, or four-word clusters, relating to a specific policy term being discussed. We believe these n-grams, or sets of words, will provide an even more precise picture of the text content, as opposed to individual terms, and allow us to find underlying patterns and linkages within text more precisely.

References

- Benchimol, J., Kazinnik, S., Saadon, Y., 2020. Communication and transparency through central bank texts. Paper presented at the 132nd Annual Meeting of the American Economic Association, January 3-5, 2020, San Diego, CA, United States.
- Bholat, D., Hans, S., Santos, P., Schonhardt-Bailey, C., 2015. Text mining for central banks. No. 33 in Handbooks. Centre for Central Banking Studies, Bank of England.
- Blei, D. M., Ng, A. Y., Jordan, M. I., 2003. Latent Dirichlet allocation. *Journal of Machine Learning Research* 3, 993–1022.

- Bruno, G., 2017. Central bank communications: information extraction and semantic analysis. In: Bank for International Settlements (Ed.), *Big Data*. Vol. 44 of IFC Bulletins chapters. Bank for International Settlements, pp. 1–19.
- Correa, R., Garud, K., Londono, J. M., Mislant, N., 2020. Sentiment in central banks' financial stability reports. Forthcoming in *Review of Finance*.
- Feinerer, I., Hornik, K., Meyer, D., 2008. Text mining infrastructure in R. *Journal of Statistical Software* 25 (i05).
- Laver, M., Benoit, K., Garry, J., 2003. Extracting policy positions from political texts using words as data. *American Political Science Review* 97 (02), 311–331.
- Loughran, T., McDonald, B., 2011. When is a liability not a liability? Textual analysis, dictionaries, and 10-Ks. *Journal of Finance* 66 (1), 35–65.
- Wickham, H., 2014. Tidy data. *Journal of Statistical Software* 59 (i10), 1–23.