mineR: An R Package for Fuzzy Key Term Identification and Quantification in Natural Language

Christopher B. Cole*
Lancaster University, University of Ottawa
chris.c.1221@gmail.com

Sejal Patel
Centre for Addiction and Mental Health

Jo Knight
Lancaster University, Centre for Addiction and Mental Health

June 17, 2016

Abstract

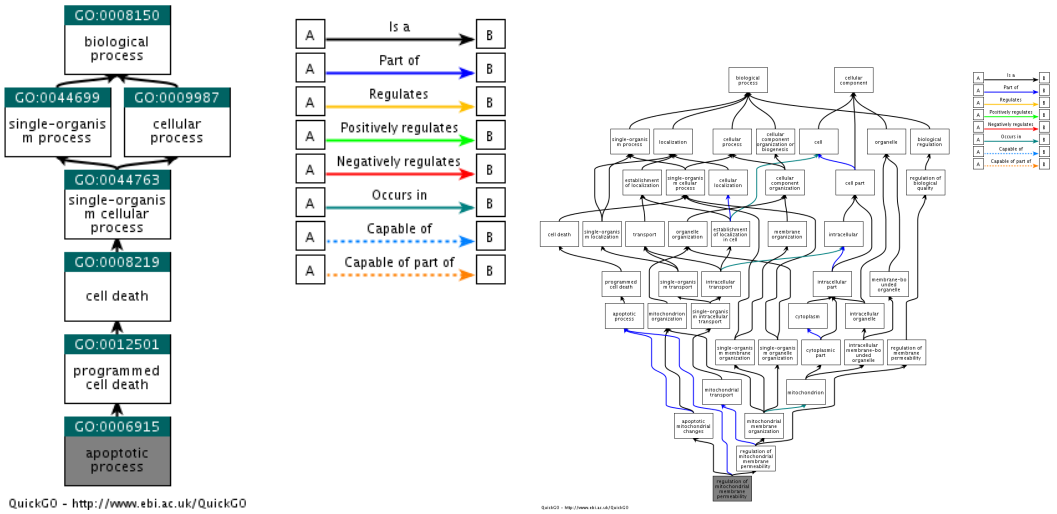
Recent growth in the scale and scope of large ontologies has prompted the development of computational methodologies which can best use structured information to further human understanding. However, using structured "terms" to mine the academic literature has proved difficult; previous efforts have neglected key concepts in natural language processing and efficient computation. In this article we present mineR, an R package capable of identifying co-occuring units within ontological terms to variable confidence, strict quality control, and additional features. mineR is released on Github and allows researchers use information from ontologies to extend and improve text mining in their field.

I. Overview

i. Introduction

i.1 Motivation

As the size and complexity of publically available data steadily increases, researchers are often unable to efficiently use unstructured information to advance human knowledge. In fields ranging from biology to linguistics, efficient processing and analysis of unstructured data has become a major challenge and source of opportunity, specifically in the areas of natural language processing (NLP) and information extraction (IE). These challenges have been especially apparent when analyzing large bodies of information for relatively complex thematic structures such as those represented in large ontologies. One such example is Gene Ontology (GO), a platform which allows computational biologists to describe gene function in terms of a structured vocabulary. These terms's complexity scales with their specificity, and thus to perform precision analysis of literature using GO terms involves the analysis of "terms", or structured phrases, which may be upwards of 10 words long using domain specific terminology. These phrases detail the relation between individual elements as well as the function within a larger network of overall function.



(a) *Relatively simplistic gene ontology (GO) term structure containing short and easily detected terminology.*

(b) *More complex gene ontology (GO) term structure containing highly specific and interrelated terminology.*

Figure 1: *Examples of Gene Ontology Term Structures*

For example, the GO term "apoptotic process" (GO:0006915) has a relatively simple ancestry structure which may be easily represented as shown in **Figure 1a**.

However, this structure quickly becomes unruly and complex once the specificity of a query increases. The corresponding strcture for the GO term "regulation of mitochondrial membrane permeability involved in apoptotic process" (GO:1902108) is displayed in **Figure 1b.**

A common application of this data is text mining. A researcher may wish to find common "terms" amongst a particular query on Pubmed. For example, a research group may want to identify common GO terms amonst a Pubmed Central (PMC) query for Alzheimers. PMC would return a list of publications which relate to Alzheimers, and the researcher aims to identify common or overrepresented GO terms in this corpus. The researcher would run into several problems.

1. Identifying a method to find co-occuring words which constitute a "term" and allowing the degree of confidence to vary by application.

2. Efficiently computing associations across hundreds or thousands of documents in a corpus.

3. Quantifying, summarizing, and reporting of associations in a meaninful way.

4. Applying strict quality control and canonical natural language processing filters to improve true positive rate (TPR) and reduce false positive rate (FPR).

In this article we present mineR as an R package capable of addressing these concerns and performing multi word term identification and quantification in natural langauge. We additionally develop methods for identifying and accounting for acronyms and synonyms by leveraging domain specific user knowledge.

*Corresponding author

i.2 Design Philosophy

In the framework proposed by Krauthammer & NenadiÄĞ (**Figure 2.**), mineR lies firmly within the bounds of term recognition; other tools have been designed which excel at categorization and mapping, and as such the job of mineR ends after terms have been identified, summarized, and quantified.

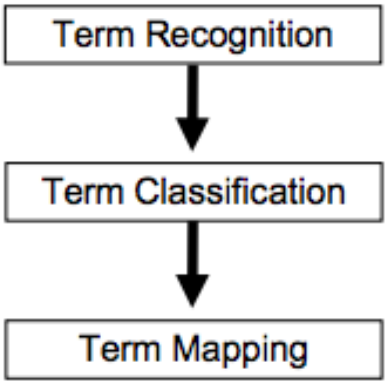


Figure 2: *Term identification framework proposed by Krauthammer & NenadiÄĞ (2004).*

The failures of recent text mining approaches in the biomedical literature have largely revolved around a large disconnect between general methodologies such as those employed by TensorFlow and domain specific knowledge which is needed to accurately understand free language. Though this is a significant issue in all fields, it is especially apparent in biology and genetics where non-dictionary terms and acronyms are the majority rather than a small portion; computational algorithms struggle without a clear cut set of rules in these cases.

mineR attempts to address these concerns by requireing human knowledge and domain specific information to function. As an example, the strength and flexibility of mineR scales with the amount of time invested by the user constructing appropriate lists of synonyms, acronyms, and negations. Though this requires a larger time investment up front, the benefits are large. We hope that these lists of synonyms, acronyms, and negations will become public knowledge and posted on public databases such as our Github project site.

i.3 Typical Usage

A typical analysis conducted with `mineR` is presented in supplemental methods 1: Basic Usage. However, here we present the rationale for any typical analysis.

Firstly a user must identify and supply a list of terms which they are interested in analysing. For gene ontology based analyses, these may be found online. For others, users will have to defer to their specific area of expertise. Examples include mining publically available government documents for long program names, mining publically available literature for thematic expressions, and mining social media such as Twitter for ideas (“Rooney scored against Real Madrid”).

Alongside this list of terms, the user must supply a single document or a corpus of documents to mine. These documents represent the sample space of the analysis.

From this, `mineR` will identify:

- Term's Frequency
- Places within document where the term was identified
- A term's relative importance

We note that `mineR` is usually, though not always, incompatible with latent semantic analysis because a large portion of its utility derives from breaking LSA's assumption that similar words appear in similar texts. For a large body of academic text this is not the case.

i.4 Algorithm

In general, the algorithm consists of the following steps.

1. Open connections to desired text, term list, and any other auxillary information like synonyms.
2. Split text into n chunks to optimize processing.
3. (Optionally distribute n chunks over k computational cores).
4. Perform quality control (as detailed in iv.)
5. Construct term document matrix (TDM) of text, terms, and synonyms.
6. Account for synonyms and acronyms by adjusting TDM of terms.
7. Match counts from text TDM and term TDM.
8. Apply thresholding.
9. (Optionally summarize over multiple texts in a corpus).
10. Summarize, format, and return results of associations.

To go into slightly more detail, connections to the text (or corpus) being mined are opened and converted from PDF or XML to a flat text document and stored within R as a vector, of which each element is a sentence delimited by a terminal punctuation mark (`.!?`). This vector has its whitespace cleared (a vestige from page breaks in the conversion process) and quality control measures are applied on the free text in order to reduce false positives and increase the discriminative power. Once the quality control measures (detailed in iv.) have been applied, a term document matrix (TDM) is constructed using the `tm` and `nlp` packages. Presently no transformations are applied to the TDM in order to alleviate any implicit assumptions on the goals of the term identification, though this will be the subject of future development and optional algorithms. An example of a TDM is displayed in **Figure 3.** This TDM is split into n chunks to facilitate either parrelell (through `doParrelell` and `mcSnow`) or reduced serial computation time.

	I	like	hate	databases
D1	1	1	0	1
D2	1	0	1	1

Figure 3: *Example of a Term Document Matrix (TDM) for two sentences. "D1": I love databases, "D2": I hate databases.*

After the term document transformation has been applied to the text, a similar transformation is applied to both the term list and the any synonyms which are presented to mineR. This is in order to standardize the quality control procedures to faciliate comparissons.

We then merge the TDMs of the terms and the text to create a smaller TDM only looking at the specific words found in the term list. We augment this term list to indicate not only if a word in a term is present, but also if a synonym of that word is found.

We then filter this matrix to include only the matches for one single term. We then take the row sums to be the number of words matched in a particular sentence from that specific term. We denote this as s_i. From this, we must decide is S_i is large enough to declare that a particular term t_i is found in a particular sentence s_i.

This is accomplished through a user defined discontinuous step function which we call the acceptance function. By default we have defined the acceptence function a as a function of n, the number of non-trivial words in a term such that

$$a(n) = \begin{cases} n, & 1 < n \leq 4 \\ n-1, & 5 < n < 8 \\ n-2, & 9 < n < 10 \\ 10, & n > 10 \end{cases}$$

If the number of words from the term present in a particular sentence exceeds $a(n)$, we call it a hit and say that the term was identified in that particular sentence.

To summarize, we count up all sentences which have had a "hit" identified and diaplay this information collapsed over terms. This information may be meta analyzed over a corpus or between corpi for comparitive analyses.

i.5 Quality Control

`mineR` incorporates many standard quality control procedures common to many NLP analysis pipelines. These include:

- Case standardization
- Punctuation elimination (with exception cases)
- Numerical elimination (with exception cases)
- Removing of Stop Words (for a given language, or standard english by default)
- Consistent word stemming
- Whitespace elimination and consideration
- Term Document Matrix (TDM) creation and reporting and frequency reporting

Given an input document or corpus of documents in PDF form and a text encoding format, `mineR` will convert documents in PDF form to plain text. If the user is unsure of formatting, `mineR` will by default guess at the formatting. Preliminary QC is performed on the documents through case standardization, punctuation elimination, numerical elimination, and the removal of stop words. The document is then stemmed, and white space is eliminated.

A similar process is conducted on the input list of terms, given one per line with unix return characters in a specified plain text document. The terms are then standardized to match the format of text in the document by the above mentioned steps.

i.6 Computational Considerations

As the number of terms increases, the computational time that the algorithm wil require increases expodentially (**Figure 4.**).

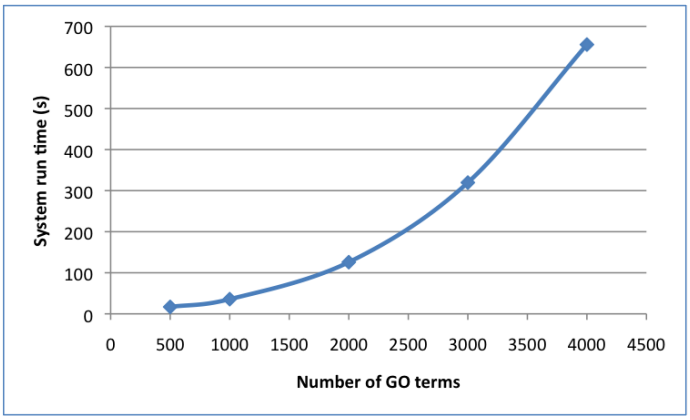


Figure 4: *`mineR` run time to completion in seconds performed on a MacBook Pro OSX v10.9.5 with a 2.66 GHz Intel Core 2 Duo processor and 4 GB 1067 MHz DDR3 RAM.*

Users are encouraged to take advantage of this behaviour and chunk terms into batches of 100-500 terms to make computational time reasonable. `mineR` includes an option to guide the user through this process, see `?mineR::term_chunk()` for details.

ii. Implimentation and Architechture

`mineR` is written in native R scripts with no compilation of other languages. It heavily uses the following packages:

1. `tm`
2. `Rcpp`
3. `dplyr`
4. `SnowballC`
5. `foreach`

Known functional versions of these libraries have been included in a `packrat` personalized CRAN instanced bundled along with `mineR`.

`mineR` contains one overarching function `mineR::mineR()` which takes two mandatory arguements, `doc` and `terms`, containing a .pdf or .txt file and a `\n` delimited list of terms to be identified. A brief example is given below in Usage Instructions and a more complete runthrough with examples and explaination is given in the "Basic Usage" vignette, also supplemental methods 1.

`mineR` only uses one core by default, but may be multi corred by setting `mc.cores = n` where *n* is the number of cores you want to use. This will additionally import `doMC` and register the local parrellel backend. Keep in mind that your milage may vary with parrellel computing depending on the size of your data set; for small sets of terms and/or small files, parrellizing may

actually slow down computational speeds due to the relatively quick iterations performed and the increased load of distributing and balancing jobs over the backbone.

mineR is designed to be used in batch operations, however for smaller scale or pilot testing, an interaction version may be used, in which the program guides the user through the logic and process of designing their terms and synonyms. This may be performed by giving ``interactive'' to any of lims, syn or acro, which will call mineR::make.lims, mineR::make.syn, and mineR::make.acro respectively.

Optionally, a log file may be written to the path specified to the log log = ``path'', or if log = TRUE then mineR will write to mineR.txt in the current working directory.

A full descriptions of arguements for the mineR is given in **Table 1**.

iii. Quality Control

mineR impliments several layers of quality control. Firstly, we bundle with mineR a packrat personalized CRAN instance to control package dependencies and ensure that functions will work as expected. Unit testing is conducted on essential functions through the testthat package, with code coverage estimated at 57% by covr at time of writing.

Additionally, continuous integration is tested for each new release by Travis CI on OSX and linux, and through Appveyor for windows.

Additionally, users may run

```
mineR::mineR(test = TRUE)
```

with no additionally arguements to verify the complete function of the package. The function will return TRUE if all is working correctly, and FALSE if there are any errors. An attempt will be made to show at which portion the test failed.

II. Availability

III. Reuse Potential

Typical and expected use of mineR is in the mining of biomedical literature, specifically through Gene Ontology terms. However, we see potential for this software outside of biology in fields such as the humanities and social media text mining. We believe that the identification of key terms is a fairly general process which may be applied in many areas, many of which we are unaware. Additionally, the code contained in this package may be included in different applications and software to incorporate the method into new areas.

IV. References

Add in references

Notes