What is the optimal topic-modelling workflow with MALLET?

Asked 8 years, 7 months ago Modified 8 years, 7 months ago

Viewed 2k times Part of R Language Collective

4

Introduction

I'd like to know what other topic modellers consider to be an optimal topic-modelling workflow all the way from pre-processing to maintenance. While this question consists of a number of sub-questions (which I will specify below), I believe this thread would be useful for myself and others who are interested to learn about best practices of end-to-end process.

Proposed Solution Specifications

I'd like the proposed solution to preferably rely on **R** for text processing (but **Python** is fine also) and topic-modelling itself to be done in **MALLET** (although if you believe other solutions work better, please let us know). I tend to use the `topicmodels` package in `R`, however I would like to switch to `MALLET` as it offers many benefits over `topicmodels`. It can handle a lot of data, it does not rely on specific text pre-processing tools and it appears to be widely used for this purpose. However some of the issues outline below are also relevant for `topicmodels` too. I'd like to know how others approach topic modelling and which of the below steps could be improved. Any useful piece of advice is welcome.

Outline

Here is how it's going to work: I'm going to go through the workflow which in my opinion works reasonably well, and I'm going to outline problems at each step.

Proposed Workflow

1. Clean text

This involves removing punctuation marks, digits, stop words, stemming words and other text-processing tasks. Many of these can be done either as part of term-document matrix decomposition through functions such as for example `TermDocumentMatrix` from `R`'s package `tm`.

Problem: This however may need to be performed on the text strings directly, using functions such as `gsub` in order for MALLET to consume these strings. Performing in on the strings directly is not as efficient as it involves repetition (e.g. the same word would have to be stemmed several times)

2. Construct features

In this step we construct a term-document matrix (TDM), followed by the filtering of terms based on frequency, and TF-IDF values. It is preferable to limit your bag of features to about 1000 or so. Next go through the terms and identify what requires to be **(1) dropped** (some stop words will make it through), **(2) renamed** or **(3) merged** with existing entries. While I'm familiar with the concept of stem-completion, I find that it rarely works well.

Problem: (1) Unfortunately `MALLET` does not work with TDM constructs and to make use of your TDM, you would need to find the difference between the original TDM -- with no features removed -- and the TDM that you are happy with. This difference would become stop words for MALLET. **(2)** On that note I'd also like to point out that feature selection does require a substantial amount of manual work and if anyone has ideas on how to minimise it, please share your thoughts.

Side note: If you decide to stick with `R` alone, then I can recommend the `quanteda` package which has a function `dfm` that accepts a `thesaurus` as one of the parameters. This thesaurus allows to to capture *patterns* (usually regex) as opposed to words themselves, so for example you could have a pattern `\\bsign\\w*.?ups?` that would match `sign-up`, `signed up` and so on.

3. Find optimal parameters

This is a hard one. I tend to break data into test-train sets and run cross-validation fitting a model of `k` topics and testing the fit using held-out data. Log likelihood is recorded and compared for different resolutions of topics.

Problem: Log likelihood does help to understand how good is the fit, but **(1)** it often tends to suggest that I need more topics than it is practically sensible and **(2)** given how long it generally takes to fit a model, it is virtually impossible to find or test a grid of optimal values such as iterations, alpha, burn-in and so on.

Side note: When selecting the optimal number of topics, I generally select a range of topics incrementing by 5 or so as incrementing a range by 1 generally takes too long to compute.

4. Maintenance

It is easy to classify new data into a set existing topics. However if you are running it over time, you would naturally expect that some of your topics may cease to be relevant, while new topics may appear. Furthermore, it might be of interest to study the lifecycle of topics. This is difficult to account for as you are dealing with a problem that requires an unsupervised solution and yet for it to be tracked over time, you need to approach it in a supervised way.

Problem: To overcome the above issue, you would need to **(1)** fit new data into an old set of topics, **(2)** construct a new topic model based on new data **(3)** monitor log likelihood values over time and devise a threshold when to switch from old to new; and **(4)** merge old and new solutions somehow so that the evolution of topics would be revealed to a lay observer.

Recap of Problems

- String cleaning for `MALLET` to consume the data is inefficient.
- Feature selection requires manual work.
- Optimal number of topics selection based on LL does not account for what is practically sensible
- Computational complexity does not give the opportunity to find an optimal grid of parameters (other than the number of topics)
- Maintenance of topics over time poses challenging issues as you have to retain history but also reflect what is currently relevant.

If you've read that far, I'd like to thank you, this is a rather long post. If you are interested in the suggest, feel free to either add more questions in the comments that you think are relevant or offer your thoughts on how to overcome some of these problems.

Cheers

R python r text-mining lda mallet

Share

Improve this question

Follow

edited May 25, 2016 at 6:44

asked May 23, 2016 at 8:52

IVR

1,838 ● 3 ● 24 ● 47

MALLET does provide k-fold cross validation through the mallet train-classifier command. Do you know if you can select a topic-modeling as a trainer? – merhoo Apr 17, 2019 at 17:35

1 Answer

Sorted by: Highest score (default)

4

Thank you for this thorough summary!

As an alternative to `topicmodels` try the package `mallet` in R. It runs Mallet in a JVM directly from R and allows you to pull out results as R tables. I expect to release a new version soon, and compatibility with `tm` constructs is something others have requested.

To clarify, it's a good idea for *documents* to be at most around 1000 tokens long (*not* vocabulary). Any more and you start to lose useful information. The assumption of the model is that the position of a token within a given document doesn't tell you anything about that token's topic. That's rarely true for longer documents, so it helps to break them up.

Another point I would add is that documents that are too short can also be a problem. Tweets, for example, don't seem to provide enough contextual information about word co-occurrence, so the model often devolves into a one-topic-per-doc clustering algorithm. Combining multiple related short documents can make a big difference.

Vocabulary curation is in practice the most challenging part of a topic modeling workflow. Replacing selected multi-word terms with single tokens (for example by swapping spaces for underscores) before tokenizing is a very good idea. Stemming is almost never useful, at least for English. Automated methods can help vocabulary curation, but this step has a profound impact on results (much more than the number of topics) and I am reluctant to encourage people to fully trust any system.

Parameters: I do not believe that there is a right number of topics. I recommend using a number of topics that provides the granularity that suits your application. Likelihood can often detect when you have too few topics, but after a threshold it doesn't provide much useful information. Using hyperparameter optimization makes models much less sensitive to this setting as well, which might reduce the number of parameters that you need to search over.

Topic drift: This is not a well understood problem. More examples of real-world corpus change would be useful. Looking for changes in vocabulary (e.g. proportion of out-of-vocabulary words) is a quick proxy for how well a model will fit.

Share Improve this answer Follow

answered May 24, 2016 at 14:27

David Mimno
1,901 ● 8 ● 7

Hi David, good to hear from you, I'm familiar with some of your work. I've tried the `mallet` package in R, however I prefer to use the command line tool instead, because it gives you more control. Since I typically run R through Rstudio, I find that shelling out to external programs (e.g. Stanford Sentiment Analytics tool) through R can crash Rstudio if they take too long to compute. – IVR May 25, 2016 at 6:57

I certainly agree that when your documents contain few tokens, it isn't worthwhile running a topic model over your corpus and I can understand why that might also be the case when you have too many tokens per document, although I have never played with such corpora. In my thread however, I was referring to the size of the vocabulary after removing stop words, infrequent words as well as too frequent words. Do you have any views on what size your vocabulary should be? – IVR May 25, 2016 at 7:07

With regards to finding the right number of topics, I also find that in practice most diagnostics tend to be abandoned in favour of the solution that makes most sense upon visual inspection. As for topic drift -- sounds like a very sensible idea to monitor the proportion of out-of-vocabulary terms, I haven't thought about this, but I will keep it in mind. – IVR May 25, 2016 at 7:13

Lastly, do you have recommendations with regard to loading text into MALLET? Specifically, how do you approach vocabulary curation? Would you do the following: (1) create a TDM in R. (2) filter out stop words, infrequent / very frequent words. (3) manually go over the vocabulary and construct regex patterns that would group your tokens (e.g. "sign_up" = "\\bsign\\w*ups?") and expand your stop word list. (4) run a for-loop on your original text grouping patterns into tokens and removing everything else. (5) finally load the text into MALLET. Do these steps sound reasonable? – IVR May 25, 2016 at 8:16