Rick Suggs
University of Illinois Urbana-Champaign
CS410 Fall 2022 - Text Information Systems
Technology Review
10-30-2022

# Gensim versus Tomoto

A Comparison between Two Topic Modeling Libraries

## Background

Topic modeling refers to deriving probabilistic topic models from an existing collection of documents. A topic model is a probabilistic distribution of topics. Topic modeling enables us to find what topics are present in documents of unstructured text data.

## Motivation

As technologists, we are motivated to solve problems using the most appropriate tools available. Discovering existing tools that make complex tasks tenable can be an exciting experience. Those new to the field of text mining and analysis can use these tools to more fully understand the theoretical foundations. At the same time, it can be gratifying to gain insight by processing an enormous set of documents to discover interesting things about the data. Documenting a comparison of tools can also help other practitioners decide what to use without spending time learning multiple tools.

## Procedure

I set out to perform a systematic comparison of the libraries; by using them to analyze the topics of the same corpus. Both of the libraries support training the Latent Dirichlet Allocation (LDA) topic model, so I used each library to train a model on the same set of test documents. The test corpus was a set of 1000 preprocessed Wikipedia articles available to download on Tomoto's documentation site.[14] I also evaluated each library on the quality of documentation, library features, development experience, and API design. I also compared the topic modeling results. Although the parameters accepted by the LDA model for each library are slightly different, I attempted to use similar parameters. I configured each library to analyze 5 topics.

## Gensim

Gensim is self described as 'topic modeling for humans'. According to the online documentation, Gensim's main features are training large scale models, representing text as semantic vectors, and finding semantically related documents.[11] The project's mission is to help NLP practitioners try out topic

modeling algorithms on large datasets, and allow researchers to prototype new algorithms.[5] Gensim is a primarily written in pure Python.

Gensim's approach to performance focuses on efficient memory usage. When processing large datasets, it streams each document one at a time from the hard disk, rather than loading the entire collection into memory. It also eliminates any vector values of zero, as a space saving measure.[1]

Topic models often take a long time to optimize, or converge. Gensim supports serialization of topic models using four serialization formats: Market Matrix, Joachim's SVMlight, Blei's LDA-C, GibbsLDA++.[2] Gensim can save trained topic models to a file and reload them efficiently when they are needed.

Gensim support the following model algorithms:[4]

- TF-IDF
- Latent Semantic Indexing
- Random Projections
- Latent Dirichlet Allocation
- Hierarchical Dirichlet Process.

It supports two implementations of the LDA topic modeling algorithm; one for single core processing, and one for multicore processing.[12] Gensim's LDA and LSA implementations also support distributed processing. Multiple physical machines can work in parallel to more efficiently process massive data sets.[13]

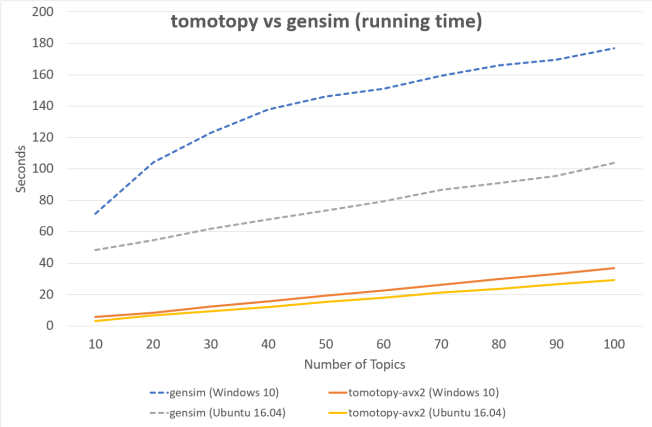
Here is an example of Python code for training LDA models using both the single core and multicore algorithms.

```
import logging
from gensim import corpora
from gensim import models
import pprint

logging.basicConfig(
    format="%(asctime)s : %(levelname)s : %(message)s", level=logging.INFO
)

dictionary = corpora.Dictionary()
lines = [line.strip() for line in open("enwiki-stemmed-1000.txt", encoding="utf-8")]
docs = [line.split() for line in lines]

dictionary.add_documents(docs)
dictionary.filter_extremes(no_below=0, no_above=1, keep_n=29327)
dictionary.filter_n_most_frequent(5)

# Bag-of-words representation of the documents.
corpus = [dictionary.doc2bow(doc) for doc in docs]

# Make an index to word dictionary.
temp = dictionary[0]  # This is only to "load" the dictionary.
id2word = dictionary.id2token

model = models.LdaMulticore(
    corpus=corpus,
    id2word=id2word,
    chunksize=2000,
    iterations=400,
    num_topics=5,
```

```
    passes=20,
    eval_every=None,
    random_state=42,
)

pprint.pprint(model.print_topics())

model = models.LdaModel(
    corpus=corpus,
    id2word=id2word,
    chunksize=2000,
    alpha="auto",
    eta="auto",
    iterations=400,
    num_topics=5,
    passes=20,
    eval_every=None,
    random_state=42,
)

pprint.pprint(model.print_topics())
```

# Tomoto

Tomoto's focuses on maximizing performance while processing large collections. As such, the core library is written in C++ and uses vectorization of modern CPUs for maximizing speed. In fact, the library's documentation site goes into some detailed comparisons between the performance of Gensim and Tomoto.[7]




example performance comparison (7)

I was excited to find that both Python and Ruby bindings also exist for Tomoto. The Ruby bindings follow the same API as the Python library.[6]

Tomoto also supports serialization of trained topic models. However, the documentation does not specify what serialization format is used. As such, it is unknown whether there is any interoperability between Tomoto and other topic modeling libraries.[7]

Tomoto supports a wider range of topic modeling algorithms than Gensim, including: [7]

- Latent Dirichlet Allocation
- Labeled LDA

- Partially Labeled LDA
- Supervised LDA
- Dirichlet Multinomial Regression
- Generalized Dirichlet Multinomial Regression
- Hierarchical Dirichlet Process
- Hierarchical LDA
- Multi Grain LDA
- Pachinko Allocation
- Hierarchical PA
- Correlated Topic Model
- Dynamic Topic Model
- Pseudo-document based Topic Model

Here is an example of Ruby code for training LDA models using Tomoto.

```
require 'tomoto'

mdl = Tomoto::LDA.new(tw: :one, min_cf: 3, rm_top: 5, k: 20, seed: 42)
File.foreach("enwiki-stemmed-1000.txt") do |line|
  ch = line.strip.split(/[[:space:]]+/)
  mdl.add_doc(ch)
end
mdl.burn_in = 100
mdl.train(0)

start_time = Time.now

100.times do |i|
  mdl.train(10)
end

end_time = Time.now
elapsed_time = end_time - start_time

puts "Elapsed Time: #{elapsed_time}"
puts mdl.summary(topic_word_top_n: 10)
```

Here is a comparison of the topics derived and elapsed time by each library. Interestingly, the Gensim LdaMulticore model did not perform more efficiently than the Gensim LdaModel.

| **Gensim Multicore LDA** | **Gensim Single Core LDA** | **Tomoto LDA** |
|---|---|---|
| Elapsed Time: 81.20s | Elapsed Time: 77.09s | Elapsed Time: 45.52s |
| work form may includ art mani acid studi languag book | work form includ may art languag mani acid studi book | work one centuri name time church languag book art god |
| system number comput includ state time design oper appl program | system number comput includ state time oper design appl program | english player politician footbal author day actor french german produc |
| american player english footbal politician author actor french produc academ | american player english footbal politician author actor french produc academ | state nation citi year new govern war countri unit includ |

| | | |
|---|---|---|
| state war year new would king time govern forc nation | state war year new king would time govern forc nation | film system comput design appl time includ one program would |
| citi centuri church state day area arab includ south region | citi state centuri day area church includ south region apollo | one may form number acid exampl includ differ two system |

## Conclusion

Both libraries have their advantages and disadvantages. Gensim has a higher adoption rate than Tomoto, and more recent development activity.[15] While both have sufficient documentation to understand how to use the libraries, Gensim is more comprehensive. Tomoto supports more topic modeling algorithms than Gensim. Gensim supports standardized serialization in multiple formats, meaning it can integrate with other data science libraries. Tomoto has bindings for both Python and Ruby, so it can easily be integrated into an existing Ruby application, if required. Based on this experiment and others, Tomoto is clearly the winner in terms of performance.[9] However, Gensim supports distributed processing for some algorithms, so it may be more suitable for large scale corpora. Overall, there is no clear winner between the libraries. Evaluating text mining solutions can lead to subjective conclusions. However, knowing the strengths of each solution can enable deciding what is best for practical purposes.

## References / Resources


1. Core Concepts — gensim
2. Corpora and Vector Spaces — gensim
3. Indexing by latent semantic analysis
4. Topics and Transformations — gensim
5. Similarity Queries — gensim
6. ankane/tomoto-ruby - high performance topic modeling
7. tomotopy API documentation (v0.12.3)
8. Don’t be Afraid of Nonparametric Topic Models (Part 2: Python) | by Eduardo Coronado Sroka
9. Looking for A Faster Way To Create Topic Models? Try Tomotopy! - Mariann Beagrie | Tealfeed
10. tmplot
11. Gensim: Topic modelling for humans
12. models.ldamulticore – parallelized Latent Dirichlet Allocation — gensim
13. Distributed Computing — gensim
14. tomotopy API documentation (v0.9.0) - Examples
15. GitHub - RaRe-Technologies/gensim: Topic Modelling for Humans