

# Topic Modeling In WISDOM

Yuening Hu          Jordan Boyd-Graber

Philip Resnik

University of Maryland

ynhu@cs.umd.edu, jbg@umiacs.umd.edu, resnik@umd.edu

December 1, 2011

This document summarizes work done in at the University of Maryland to extend the capability's of the WISDOM framework for exploring datasets. There were two main components to this work: facilitating the extension of WISDOM to handle Arabic, and conducting research on multilingual topic modeling with an eye toward integration of capabilities within the WISDOM framework.

With respect to the first component, Sections 1 and 2 discuss the creation of sentiment dictionaries for Arabic and the development of an NLP pipeline for Arabic suitable for use in WISDOM. Sections 3 and 4 discuss baseline monolingual topic modeling using the Mallet toolkit, and novel research on improving the scalability of topic modeling tools in a multilingual setting.

## 1 Creating Sentiment Dictionaries for Arabic

### 1.1 Background

Before describing our process for creating an Arabic dictionary, we must first describe some of the resources used to create the dictionary. The most important resource is WordNet, which is an electronic dictionary that organizes words based on meaning (rather than how the word is written, as in paper dictionaries).

WORDNET is composed of many “synsets” (synonym sets) that contain words that have the same meaning. For example, “pen” and “penitentiary” appear in the same synset representing a place where criminals are imprisoned. However, because WORDNET distinguishes the meaning of concepts, “pen” also appears in a synset concerning writing utensils.

WORDNET has been translated into many languages, including Arabic. The process of translating a WORDNET maintains the distinction of meaning across languages and links English meanings with their Arabic meanings.

### 1.1.1 Disambiguating an Existing Sentiment Lexicon

Because WORDNET is organized with respect to meaning, it enables a task known as word sense disambiguation, which is the problem of mapping an occurrence to a word to its meaning (sense). For example, given a word like “bank” determining whether it means land by a river or a financial institution.

Most sentiment lexicons do not discriminate based on meaning (or even part of speech), thus we must determine which meaning is meant by a word. Lacking context, we use the first-sense heuristic, which selects the most frequent sense associated with each word. This achieves good performance in disambiguation tasks.

## 1.2 Creating an Arabic Sentiment List

With the above background, we are now able to explain the process for creating a sentiment dictionary from English to Arabic. In this process we have used the existing Lockheed Martin sentiment dictionary, but this could be used for any existing sentiment lexicon.

This process is shown in Figure 1. First, we disambiguate the input English words, considering the first sense for every part of speech, ignoring other parts of speech. This now gives us a list of synsets, which we can look up in the Arabic WORDNET (some will not be translated), which in turn gives us a set of Arabic words, with which we can populate our sentiment dictionary.

The process is detailed in programmatic detail below:

#### 1. Input:

- English sentiment words list: positive.dic, negative.dic;
- English WORDNET 2.0;
- Arabic dictionary: each Arabic word is mapped to a synset (offset + pos) of English WORDNET 2.0;

#### 2. Extracting Arabic sentiment words:

- Script: getEng\_synsets.py, lib/wordnet.py
- Command: python getEng\_synsets.py
- Steps:
  - Read in Arabic WORDNET, which maintains a mapping from Arabic synsets to their English equivalents.<sup>1</sup>
  - Load in English WORDNET 2.0;
  - For each word in the English sentiment word list, find the synsets of these words in English WORDNET 2.0, so we get a set of synsets for each pos tag (5 pos tags);

---

<sup>1</sup>Note that offsets are version dependent; Arabic WORDNET encodes the offset as the combination of one digit of 14 and the real offset of English WORDNET 2.0 (n:1, v:2, s:3, a:3, r:4); as a result, we have 5 dictionary corresponding to each pos, and each dictionary is mapping offset to Arabic word

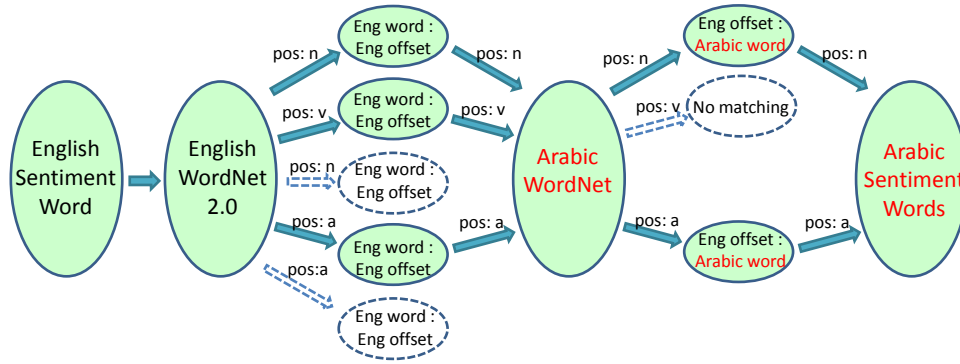


Figure 1: Extracting the Arabic sentiment word list.

- For each synset of each pos tag, search that synset in the corresponding pos dictionary. If the offset is found, output the corresponding Arabic word as one sentiment word.

### 3. Output:

- Arabic sentiment words list: arabic\_positive.dic, arabic\_negative.dic;
- Matched Arabic sentiment words list: arabic\_positive\_matched.dic, arabic\_negative\_matched.dic; for each Arabic word, the matched English word, as well as the pos and synset (offset) are also listed.

## 1.3 Caveats, Suggestions, and Future Directions

**Validation** This process produces a sentiment dictionary, but any such output should be vetted either qualitatively by a native Arabic speaker nor quantitatively using a reference corpus. This should be done before placing the system into production.

So long as the English disambiguations are correct, no errors should be introduced in the translation process. However, even though the meaning is precise, whether a word is sentiment bearing might be different across languages.

For the provided sentiment dictionary, we have used an Arabic informant to vet the sentiment list. This should be repeated for any subsequent versions of the dictionary.

**Consistency with Stemming** The provided sentiment dictionary contains base forms with vowels. Depending on when sentiment analysis is done, this may be inconsistent with how words are represented (e.g. if compared to unreduced words, terms will not match with the sentiment dictionary).

**Inclusion of Arabic-specific Words** The technique described above will only match synsets for which there is an English equivalent. Not all of the words in Arabic WORDNET, however, are mapped to English synsets.

**Removing Sense and Part-of-Speech Ambiguity** Not all senses of a word are sentiment bearing (e.g. “gross” before deductions and “gross” disgusting), and not all the parts of speech a word can be used can be sentiment bearing (e.g. “fair” the noun (carnival) and “fair” the adjective). Removing these ambiguities would allow for a more accurate determination of sentiment.

## 2 Enabling Arabic Preprocessing

This section describes the integration of Arabic in a way consistent with WISDOM’s dataflow for English documents, resulting in a demonstration (“demo”) of Arabic capabilities consistent with the WISDOM dataflow. Proof-of-concept data and code that we provided are referred to in this document.

We provide a detailed description of how to use several commercially-friendly open source tools to create the necessary components, along with guidance on how to use the same framework to create production code; we also include a brief description of several tools we explored that may be useful but whose utility may be limited by their licensing.

As we do not have access to the WISDOM API or source code, these features cannot be directly placed into WISDOM. Rather, the code snippets provided here aim as a guide to how developers would obtain the essential functionality needed by WISDOM for Arabic. To make this as simple as possible, we have isolated each functionality to individual demos.

### 2.1 Tools and License

The tools we are using are mainly the openNLP java package and the Arabic Apache Lucene java package.

- **Apache Lucence** (<http://lucene.apache.org/>). There is an extended package for Arabic analysis in this toolkit ([org.apache.lucene.analysis.ar](http://org.apache.lucene.analysis.ar)), including Arabic Normalizer and Arabic Stemmer, which are used in the demo. The license is Apache License, Version 2.0 (<http://www.apache.org/licenses/LICENSE-2.0>), which allows you to “freely download and use Apache software, in whole or in part, for personal, company internal, or commercial purposes; or use Apache software in packages or distributions that you create.”<sup>2</sup>
- **openNLP** (<http://incubator.apache.org/opennlp/>). It includes the java package for training a tokenizer, sentence detector and name entity finder. The license is also Apache License, Version 2.0.

---

<sup>2</sup><http://www.apache.org/foundation/licence-FAQ.html#WhatDoesItMEAN>

## 2.2 Demo of Arabic component capabilities for WISDOM

In the following, we describe the data we use, then how to train models (sentence detector and name entity finder), and finally details about how to do Arabic sentiment analysis and named entity detection using these models.

### 2.2.1 Data

Data are stored in the data folder of the top level WISDOM folder.

- quran-simple-clean.txt: quran data downloaded from webpage ([http://tanzil.net/wiki/Download\\_Quran\\_Text](http://tanzil.net/wiki/Download_Quran_Text));
- quran-simple-clean\_tmp.txt: remove the English notation at the very end of quran-simple-clean.txt;
- quran-simple-clean\_tmp.txt.punct: add some punctuation in quran-simple-clean\_tmp.txt, and merge some sentences into one line;
- arabic\_ner.train: a Wikipedia page about Obama; used Google Translate to translate into arabic, and “Obama” was tagged as the named entity manually.

### 2.2.2 Model Training

We need two models: sentence detector and named entity finder. We use openNLP tools to train these models. These models are stored in the mymodels folder.

- Sentence detector
  - Data: quran-simple-clean\_tmp.txt.punct
  - Command:  

```
cd apache-opennlp-1.5.1-incubating/  
bin/opennlp SentenceDetectorTrainer -encoding UTF-8 -lang ar -data ../../data/quran-simple-clean_tmp.txt.punct -model ../../mymodels/arabic-sent.bin -cutoff 0
```
  - Output: arabic-sent.model.
- Named entity finder
  - Data: arabic\_ner.train.
  - Command:  

```
cd apache-opennlp-1.5.1-incubating/  
bin/opennlp TokenNameFinderTrainer -encoding UTF-8 -lang ar -data ../../data/arabic_ner.train -model ../../mymodels/arabic-ner-person.bin
```
  - Output: arabic-ner-person.model.

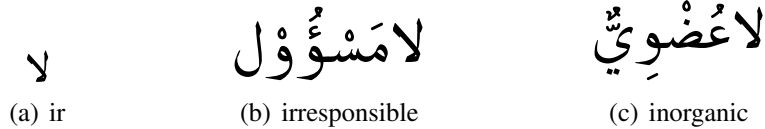


Figure 2: Arabic words

### 2.2.3 Negation

In order to detect the negations in the text, we need a negation word dictionary. We used this resource, geared toward MSA, (<http://arabic.tripod.com/Negation1.htm>) to generate the negation dictionary, presented as "negation.dic" in "mymodels" folder.

We also go beyond *lexical* negation. In addition, negation can happen via prefixes in Arabic. For example, the word shown in 2(a) is similar to the negative prefix "ir," "in," "dis," "un," "non" in English. Words with such prefix should also be detected as negation. For example, word shown 2(b) means "irresponsible" and word shown 2(c) means "inorganic."

Note that using a prefix for negation in Arabic is more common than in English, so it would be difficult to explicitly encode the appropriate polarity in the sentiment dictionary.

### 2.2.4 Sentiment Analysis

Figure 3 shows the dataflow for sentiment analysis in English WISDOM (Lockheed Martin Proprietary Information). The basic process of Arabic sentiment analysis we outline here is the same as the English sentiment analysis. Given the input text, we need to detect the sentences, tokenize the sentence, and then check the sentiment of each token. The process is shown in Figure 4. Notice that when the dictionaries are loaded, these words must be preprocessed using the same normalizer and stemmer as for the test data.

In Figure 4, two most important steps are to determine the sentiment of a word and check this word is a negation word or not. After normalizing and stemming, we first check whether this word is contained in the positive or negative dictionaries and then tag it sentiment as either "POS," "NEG" or "NULL." Then we check whether this word is contained in the negation dictionary or starts with the negation prefix mentioned above. If yes, we label this word with "NGTN." If this word is not negation, we only output the sentiment of this word. If it is negation, the output format is "sentiment-NGTN." (Some examples of output can be found in the results/arabic\_st.test.result). The details of this process are as follows:

- Load Arabic dictionaries, tokenizer, and sentence detector.
- Use sentence detector to detect sentences in the input text.
- For each sentence,
  - Tokenize this sentence by whitespace, removing punctuation.
  - For each token,

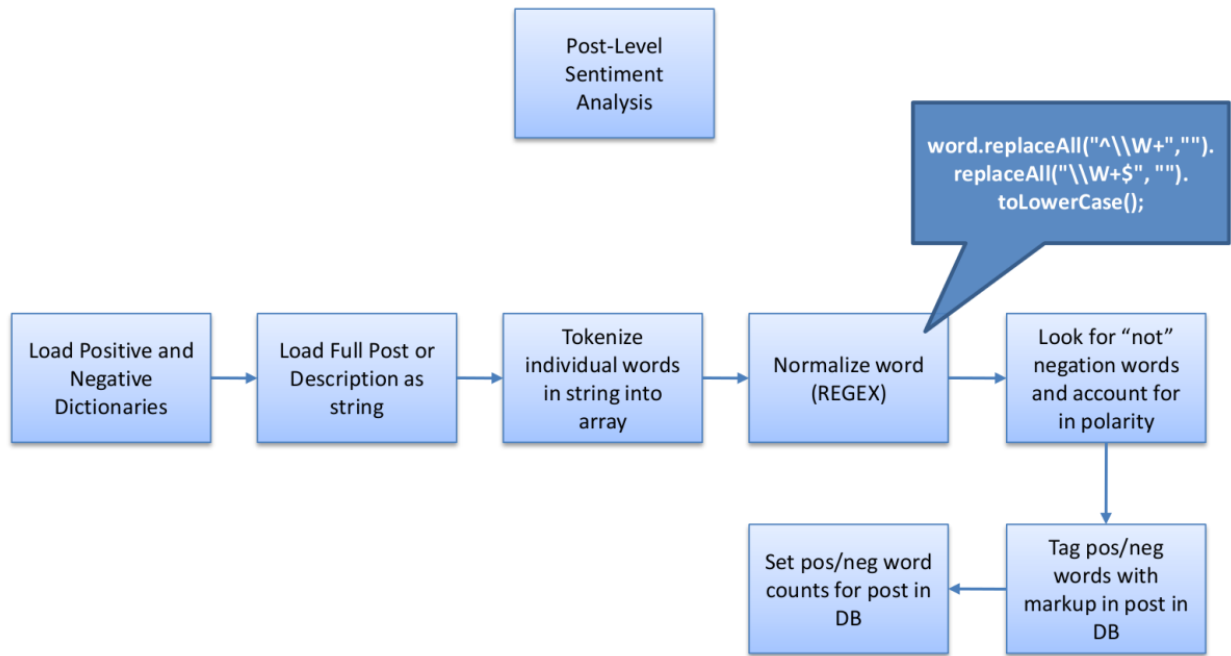


Figure 3: English Sentiment Analysis (From Lockheed Martin)

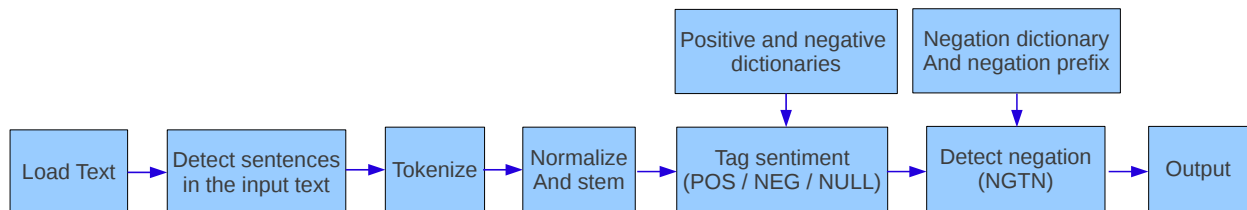


Figure 4: Arabic Sentiment Analysis

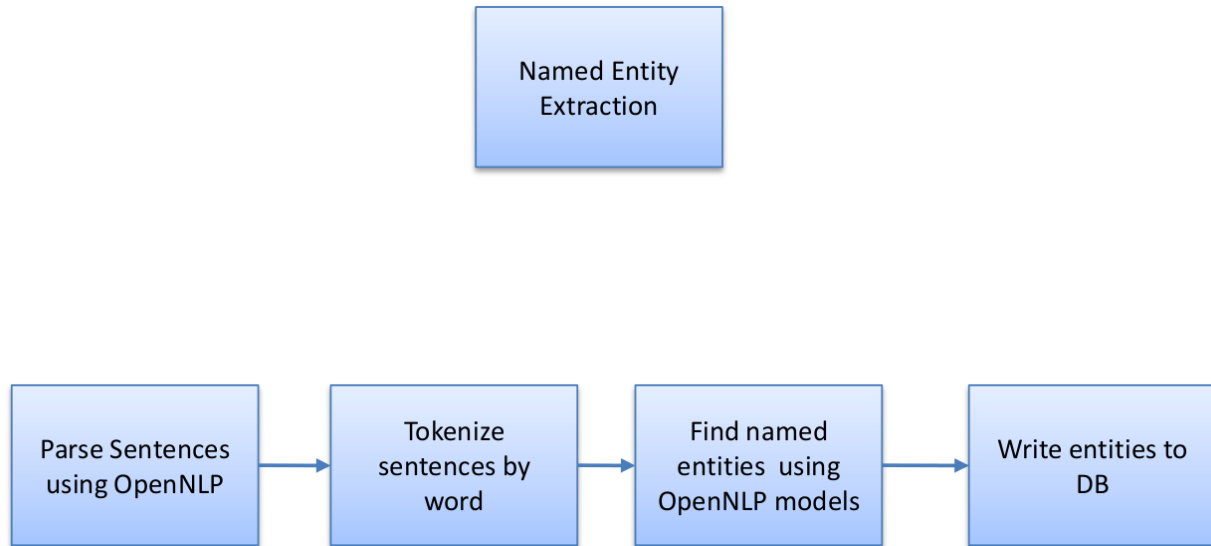


Figure 5: English named entity recognition (From Lockheed Martin)

- \* Normalize the token.
  - \* Stem the token, and get prefix, stemmed word, suffix of this token.
  - \* Determine the sentiment of this token: if the stemmed word is contained in positive dictionary, tag it as "POS"; else if the stemmed word is contained in negative dictionary, tag it as "NEG"; else tag it as "NULL".
  - \* Determine whether this token is negation or not: if the stemmed word is contained in negation dictionary or starts with negative prefix, it is a negation.
  - \* Return the tag of this token: if this word is not negation, we only output the sentiment of this word; else, the output format is "sentiment-NGTN".
- Output the result of this sentence.

### 2.2.5 Arabic Named Entity Recognition

Arabic named entity recognition is identical to the English pipeline, which is shown in Figure 5. As a result, in Arabic named entity recognition, we will first detect the sentences, tokenize the sentence, and then extract the name entities in these tokens.

Arabic named entity recognition can be described as follows:

- Load tokenizer, sentence detector, and named entity finder.
- Use sentence detector to detect sentences in the input text.
- For each sentence,

- Tokenize the sentence as before
- Find name entities in these tokens
- Output the result of this sentence

### 2.2.6 Quotation extraction

We also provide a function in the demo package “wisdom\_demo/quotationextraction.” It takes a string of Arabic text and returns an array of strings representing all quotations. There are two common ways to represent quotations in Arabic: either ”؛ ” (popular in areas with more contact with France) or “ ”, which are extracted by the provided code. However, not all quotations are explicitly demarcated (as that is a western convention), so detecting undemarcated quotations is an open research question.

### 2.2.7 How to run the process

- Demonstration of sentiment analysis pipeline
  - Input: ../results/arabic\_st.test
  - Command:
 

```
cd wisdom_demo/
java -classpath bin:lib/* sentimentanalysis.Driver ../results/arabic_st.test
```
  - Output: ../results/arabic\_st.test.result
- Demonstration of named entity pipeline
  - Input: ../results/arabic\_ner.test
  - Command:
 

```
cd wisdom_demo/
java -classpath bin:lib/* nameentity.Driver ../results/arabic_ner.test
```
  - Output: ../results/arabic\_ner.test.result
- Demonstration of quotation extraction
  - Input: ../results/arabic\_quote.test
  - Command:
 

```
cd wisdom_demo/
java -classpath bin:lib/* quotationextraction.Driver ../results/arabic_quote.test
```
  - Output: ../results/arabic\_quote.test.result

## 2.3 Guidance on Creating Production Code

In order to integrate the components demonstrated above into the WISDOM application for production use applications, the models we have used for tokenizer, sentence detector, and named entity finder, as well as the two sentiment dictionaries, are very important.

Here are some suggestions for refining the process we followed in demonstrating how to create these components:

- The training data for sentence detection is very important. It should reflect the domain of the target sentences as closely as possible. Because we were restricted to public domain works (e.g. not Web text), the training data we used will most likely not provide as good a model as real Web data with known sentence boundaries.
- The training data for named entity detection provided here **are for illustrative purposes only**. Named entity detection requires a real dataset with consistent, reliable annotation, in a domain similar to the target data.
- A more data-driven approach to negation would use a treebank with negation annotated. As this is not available for commercial uses, this was not practical for this exercise. If a strict dictionary-based negation method is used, a more complete grammar could improve the system.<sup>3</sup>
- For further caveats related to sentiment, please refer to the previous deliverable.

## 2.4 Other Tools Explored

In addition to openNLP and Apache Lucene, we also investigated other tools, which might offer better performance but are unsuitable because of license issues. We list them here for reference.

- **MADA+TOKAN** (<http://www1.ccls.columbia.edu/cadim/MADA.html>): this tool can extract the prefix, suffix, and root of a raw word. This tool can tokenize the training data and also stem text. Notice that the output is in Buckwalter, a widely used Arabic transliteration scheme. Extra tools are necessary to convert Buckwalter back to Arabic script (see below). In order to use this toolkit, users need to accept the “Non-Commercial Software License Agreement.” ([http://www1.ccls.columbia.edu/MADA/MADA\\_license.html](http://www1.ccls.columbia.edu/MADA/MADA_license.html)).
- **AraMporph** (<http://www.nongnu.org/aramorph/>): a java package that includes an Arabic stemmer and tokenizer. Similarly to MADA+TOKAN tools, it can extract prefix, suffix and stemmed word given a token. This tool is governed by “GNU General Public License.” (<http://www.gnu.org/copyleft/gpl.html>).
- **Jqurantree** (<http://corpus.quran.com/java/>): a java package which can convert Arabic to BuckWalter and BuckWalter back to Arabic. There are also some other APIs in this package, basically for accessing and analyzing Arabic Quran. This tool is an open source project, and the Quranic Arabic Corpus is available under the GNU public license with terms of use.

---

<sup>3</sup>E.g. <http://www.archive.org/details/AReferenceGrammarOfModernStandardArabic>

## 3 Monolingual Topic Modeling in WISDOM

This document describes how to extract data from a running WISDOM installation and fit topics to the data. We describe the preprocessing, our techniques, and questions for further integration and extension.

### 3.1 From Wisdom to Topics

In order to do topic modeling on the Wisdom data, there are basically four steps: extract the data from Wisdom system, preprocessing the data (tokenize, remove the stop words, etc.), change the data into Mallet input format and then do topic modeling. We will give the details in the following subsections, while the last two steps are both contained in subsection “Topic Modeling Using Mallet Toolkit”.

#### 3.1.1 Extracting Data From Wisdom

There are two parts of data we need from the Wisdom system: (1) documents; (2) the stopword list. Both of the two parts are accessed and downloaded using the JDBC API to connect to the remote database, execute appropriate MySQL statements, and write documents to a local directory.

Extraction code is implemented in Java, contained in the Java code “code/topic\_modeling\_demo/src/downloadData.java” (call function `getData(outputdir)`). Notice it is needed to setup the host URL, username and password in the Java code “code/topic\_modeling\_demo/src/Driver.java”. The default output is,

- data/wisdom\_en\_original\_data/: the documents are contained in this folder, indexed by integer IDs.
- data/stopwords.

Currently, the text is from the field “all\_text” in table “posts,” and stopwords are from the tables “stopwords” and “items.” However, it is not clear whether we should target the column “all\_text” or “full\_post” in the “posts” table, and there is no language identification for the posts and stopwords. This will be necessary for multilingual topic modeling, our next objective.

#### 3.1.2 Preprocessing Documents

In this part, the documents are tokenized based on the trained English tokenizer in openNLP. Also, stop words are removed based on the stop words list extracted by step(1).

The preprocessing code is contained in “code/topic\_modeling\_demo/src/preProcessing.java”, and the function “tokenizeFiles” is called in the “code/topic\_modeling\_demo/src/Driver.java”. The default output of this step is in “data/wisdom\_en\_data/.”

### 3.1.3 Topic Modeling Using Mallet Toolkit

In order to do topic modeling using Mallet Toolkit, we need to first convert the documents into Mallet input format, and then train topic models. For easier usage, we have integrated the Mallet Toolkit into the demo framework.

Notice, while using the Mallet Toolkit to change the input data format, some words contained in the default Mallet stop word list will also be removed. In training topics, the number of topics, number of iterations can be specified via the command.

All the results can be found in the “output” folder. We can find the topic assignments for words in each documents in the file specified by “output-doc-topics,” and the top words of each topic can be found in “output-topic-keys” file. For the file “inferencer,” we can use that to infer the topics for test data.

## 3.2 Topic Modeling Demo

Since we integrated the four steps into one framework, it is relatively easy to use. Considering we only need to update the data sometimes, there is a control boolean flag “update-data”. The data will be downloaded and preprocessed only if this flag is true.

The command is,

- `cd code/topic_modeling_demo`
- `java -classpath bin;lib/* Driver -update-data true -data-name wisdom_en -data-folder ../data -input-folder ../input -output-folder ../output -num-topics 10 -num-iterations 10 -optimize-interval 10 -optimize-burn-in 100`

You can specify the training parameters in the above command, and all the output files are contained in the “../output” folder. Notice if you run code in windows, you can use the above command directly. If you run this code in linux system, you probably need to change “-classpath bin;lib/\*” to “-classpath bin:lib/\*”. We can provide an API interface if it is needed.

## 3.3 Remaining Questions

**Data Source** There are multiple sources from which we could extract data, e.g. “all\_text” and “full\_post”; lacking certainty, we chose to use “all\_text”. Is there a source that provides the tokenized text (this is essential for non-English documents, c.f. the discussion of Tokenization in Section ??).

**Language Identification** We cannot use WISDOM for multilingual topic modeling directly without topic identification. While we can integrate language identification as a preprocessing step, we imagine this would also be useful for WISDOM’s other components.

## 4 Improving Multilingual Topic Modeling

Moving from monolingual to multilingual topic modeling presents a number of challenges. First, we require some linguistics resources to bootstrap the connection between languages. Next, we describe a process for improving the efficiency of this sampling process so that it can be done more efficiently on a single machine.

### 4.1 Mapping Across Languages

To extend topic modeling from one language to multiple languages, there are three main problems to consider: (1) how to find matched word pairs in different languages; (2) how to handle one-to-many or many-to-many mapping in the topic modeling; (3) how to connect the matched words in different languages in topic modeling.

The first problem is relatively easy to solve, since we can always depend on existing dictionaries which have the matched word pairs. However, besides one-to-one mappings, there might exist one-to-many or many-to-many mappings, which might cause problems when we try to connect the matched words in topic modeling.

In order to solve problem (2) and (3), a tree-based prior structure is explored and applied in topic modeling [2, 1, 3, 5]. Intuitively, these techniques attempt to encode correlations between words whose meanings are shared across languages. There are many resources for building these correlations, such as dictionaries or more complicated statistical structures such as WordNets [6] (Figure 6).

Note that such techniques do not force equality or equivalence between terms that are linked in the underlying linguistic resource. The resource only encourages that the distributions are *correlated*. E.g. if we see “wunsch” in German in topic 4, the probability of “wish” in English also rises for topic 4.

The technical tool we use to accomplish this is a Dirichlet Forest. A Dirichlet Forest is a collection of Dirichlet distributions arranged in a tree. A Dirichlet Forest is used as a prior for generating *each* of our topics. A multilingual topic then becomes a tree-structured collection of multinomial distributions, where each multinomial distribution has a distribution over its children.

One nice feature of using this tree-structure-based prior is that it still retains conjugacy. As a result, we can still use Gibbs sampling for inference, which simplifies the inference process greatly. Note that each node has words in a number of languages, over which there is a multinomial distribution over all words in this matched set. The generative story is:

- For each topic  $k$  and each node in that topic’s tree  $c$  that has children, draw a multinomial distribution over all the nodes  $\pi_{k,c} \sim \text{Dir}(\beta_c)$ .
- For each topic  $k$  and each node in that topic’s tree  $l$  that does not have children, draw a multinomial distribution  $\phi_{k,l} \sim \text{Dir}(\beta_l)$  over all the words (in multiple languages) in this set. We construct the tree so that no node has both children and associated words (i.e. nodes that have words are leaf nodes)
- For each document  $d$

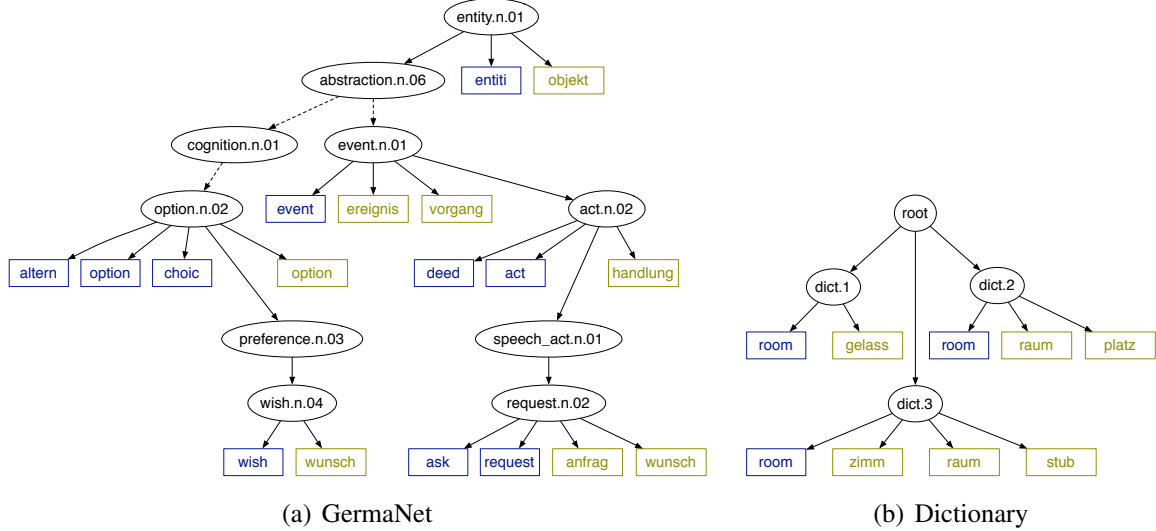


Figure 6: Two methods for constructing multilingual distributions over words. On the left, paths to the German word “wunsch” in GermaNet are shown. On the right, paths to the English word “room” are shown. Both English and German words are shown; some internal nodes in GermaNet have been omitted for space (represented by dashed lines). Note that different senses are denoted by different internal paths, and that internal paths are distinct from the per-language expression.

- First draw a multinomial distribution  $\theta_d$  to given a topic weighting for that document
- For each word  $n$  in document  $d$ ,
  - \* Draw a topic indicator  $z_{d,n} \sim \text{Mult}(\theta_d)$
  - \* Start at the root of the topic tree indexed by  $z_{d,n}$ . Choose a child from  $\lambda_{d,n,0} \sim \text{Mult}(\pi_{z_{d,n}, \text{ROOT}})$ .
  - \* Continue appending to the path  $\lambda$ :  $\lambda_{d,n,t+1} \sim \text{Mult}(\pi_{z_{d,n}, \lambda_{d,n,t}})$
  - \* Until you hit a leaf node. When you hit a leaf, node, draw a word  $w_{d,n} \sim \text{Mult}(\phi_{k,l})$ .

Notice having this tree structure, each word is associated with not only a topic but also a path, which is a set of multiple edges. Each path can be assigned with an id (e.g. via a depth first search). We use  $(i \rightarrow j)$  to denote an edge.

However, our dataset only contains words ( $w_{d,n}$  in our generative story). All of the other latent variables are completely unobserved. We discover a posterior distribution over these latent variables using posterior inference [8, 9]. In particular, we use Gibbs sampling of the topic assignments  $z$  and paths  $\lambda$ , integrating over the other latent variables ( $\theta$ ,  $\pi$ , and  $\phi$ ).

For the  $n^{\text{th}}$  word in document  $d$ , the conditional probability of that word taking on topic  $t$  and the path is  $l$  is

$$p(z_{d,m} = t, \lambda_{d,m} = l | Z_{-(d,m)}, L_{-(d,m)}) \propto (\alpha_t + n_{t|d}) \prod_{(i \rightarrow j) \in l} \frac{\beta_{i \rightarrow j} + n_{i \rightarrow j|t}}{\sum_{j'} (\beta_{i \rightarrow j'} + n_{i \rightarrow j'|t})} \quad (1)$$

where  $\alpha_t$  is the hyperparameter for topic  $t$ ;  $n_{t|d}$  denotes the frequency of topic  $t$  in  $d^{th}$  document;  $Z_{-(d,m)}$  and  $P_{-(d,m)}$  are the topic and path assignment for words other than the current one;  $\beta_{i \rightarrow j}$  is the prior for edge  $i \rightarrow j$  and  $n_{i \rightarrow j|t}$  is the number of times that edge  $i \rightarrow j$  has been seen in topic  $t$ .

## 4.2 Efficient Sampling

Previous work has explored the technique mentioned discussed in Section 4.1 [2] in detail. However, inference is difficult for larger datasets or for more complicated linguistic resources to provide correlations between languages.

Yao and Mimno et al. proposed an efficient inference for (unconstrained, monolingual) LDA [10], which is both time and memory much more efficient than the traditional LDA. We extended this technique, which they call SparseLDA, to solve the computation efficiency problem of topic models that use a Dirichlet Forest prior for topics to encourage correlations between related terms.

### 4.2.1 SparseLDA

SparseLDA rearranges the sampling equation into three terms (Equation 2). In the sampling process, SparseLDA computes the sum over all topics for each of the three items, recored as three separate bins. Additional savings can be achieved by caching the value of the first two bins rather than recomputing the value for each sampling step. As a result, sampling is much faster and the inference process has a much smaller memory footprint.

$$\begin{aligned} p(z_{d,m} = t | Z_{-(d,m)}) &\propto (\alpha_t + n_{t|d}) \frac{\beta + n_{w|t}}{\beta V + n_{\cdot|t}} \\ &\propto \frac{\alpha_t \beta}{\beta V + n_{\cdot|t}} + \frac{n_{t|d} \beta}{\beta V + n_{\cdot|t}} + \frac{(\alpha_t + n_{t|d}) n_{w|t}}{\beta V + n_{\cdot|t}} \end{aligned} \quad (2)$$

In addition, they rank the topic frequency in each document and the word frequency in each topic, especially the fast sorting can be done in  $O(N)$  time. Comparing with the normal LDA, these strategies made the inference both time and memory more efficient in the sampling process.

### 4.2.2 Efficient tree-based sampling

We extend Yao’s idea to the tree-based sampling, also by rearranging the sampling equations similarly to Equation 2. We have

$$p(z_{d,m} = t, l_{d,m} = p | Z_{-(d,m)}, L_{-(d,m)}) \propto (\alpha_t + n_{t|d}) N_{t,p} [S_p + O_{t,p}] \quad (3)$$

where  $N_{t,p}$  is the normalizer for path  $p$  in topic  $t$ ,  $S_p$  is smoothing factor for path  $p$ , and  $O_{t,p}$  is the observation for path  $p$  in topic  $t$ . The equations for computing these three factors are:

$$N_{t,p} = \prod_{(i \rightarrow j) \in p} \sum_{j'} (\beta_{i \rightarrow j'} + n_{i \rightarrow j'|t}) \quad (4)$$

$$S_p = \prod_{(i \rightarrow j) \in p} \beta_{i \rightarrow j} \quad (5)$$

$$O_{t,p} = \prod_{(i \rightarrow j) \in p} (\beta_{i \rightarrow j} + n_{i \rightarrow j|t}) - \prod_{(i \rightarrow j) \in p} \beta_{i \rightarrow j} \quad (6)$$

Then we rearrange the sampling equation as,

$$p(z_{d,m} = t, l_{d,m} = p | Z_{-(d,m)}, L_{-(d,m)}) \propto \underbrace{\frac{\alpha_t S_p}{N_{t,p}}}_{\text{Smoothing}} + \underbrace{\frac{n_{t|d} S_p}{N_{t,p}}}_{\text{Topic Beta}} + \underbrace{\frac{(\alpha_t + n_{t|d}) O_{t,p}}{N_{t,p}}}_{\text{Topic Term}} \quad (7)$$

This gives us the same three bins as in SparseLDA. We refer to these three bins as the *smoothing only* bin, the *topic beta* bin, and the *topic term* bin.

Now, the tree-based sampling makes the sampling process significantly more complex. We list additional complexities of this approach in Table 1 along with the solutions we have employed in order to address them.

These strategies successfully solved the problems and made the tree-base sampling much more efficient. The sampling process is shown in Algorithm 1.

### 4.2.3 More efficient tree-based sampling based on over-estimation

Sampling as in section 4.2.2 has already greatly improved the computation speed; however, we noticed that the computation of the smoothing-bin (which happens for every word) was taking a considerable amount of time. Observe, though, that it is possible to compute a constant upper bound of each word’s smoothing bin:

$$\text{smoothing-bin} = \frac{\alpha_t S_p}{N_{t,p}} = \frac{\alpha_t \prod_{(i \rightarrow j) \in p} \beta_{i \rightarrow j}}{\prod_{(i \rightarrow j) \in p} \sum_{j'} (\beta_{i \rightarrow j'} + n_{i \rightarrow j'|t})} \leq \frac{\alpha_t \prod_{(i \rightarrow j) \in p} \beta_{i \rightarrow j}}{\prod_{(i \rightarrow j) \in p} \sum_{j'} \beta_{i \rightarrow j'}} \quad (8)$$

Recall that sampling from a multinomial can be thought of as landing a ball in differently sized bins. Because it is expensive to compute the exact size of the smoothing only bin, we replace it by a slightly larger bin that is always bigger than the real smoothing only bin.

When, during sampling, our ball lands in the slightly larger bin, we replace the slightly larger bin with the correctly slized bin and then determine if the ball would have landed in the smoothing bin *if it had been used from the beginning*. If not, we use the correct bin, preserving the correctness of the sampling equation.

If the ball misses the (slightly over-size) sampling-only bin, then we don’t have to worry about making a mistake because the sampling-only bin is always smaller than its placeholder. Thus, whenever the ball does not hit the sampling only bin, we save on the cost of computing that bin. This algorithm is shown as Algorithm 2.

Problem	Solution
The normalizer is unique for each path, so it is different for each word. The normalizer is always the for SparseLDA.	Split the normalizer to two parts: root normalizer and remained normalizer, and cached the normalizer for each path.
When a normalizer for one path is changed, it will influence the normalizers of cousin paths (cousin paths are paths that share an edge with a given path).	Build a mapping between a node and paths containing this node. When count of a non-root node is changed, update the normalizer of all paths containing this node. If we didn't split the normalizer to two parts, since every path contains the root, we need to update all paths as long as the count for one path is changed. So it saves a large amount of computation here.
Since normalizer is always changing, the update of the first two bins take much more time than recompute them directly.	We do not, as in SparseLDA, cache the first two terms; we demonstrate that it is possible to upper bound the Smoothing Only term while still being able to do exact sampling
When compute the last bin, we need to compute all paths with <i>any</i> non-zero edges, not only non-zero paths (e.g. over all edges that are a part of the path).	Cache the count of non-zero paths by encoding the edge mask and real path count into a 32 bit integer. If an edge count is non-zero in this topic, the mask for this edge is 1, or it is 0. As a result, a path might have non-zero count even if the path never exists in this topic. Since the path mask is high bits, it is still easy to change the count of this path. The mask will only be updated when the count of an edge changed from zero to non-zero or the other way.

Table 1: Difficulties in extending the SparseLDA analysis to tree-structured priors and our solutions to these problems.

---

**Algorithm 1** FAST SAMPLING

---

```
1: for word w in this document:
2:   compute smoothing-only-bin
3:   compute topic-beta-bin
4:   compute topic-term-bin
5:   norm = sum of the three bins
6:   sample = rand()
7:   sample * = norm
8:   new-topic = -1
9:   if sample < smoothing-only-bin:
10:    sample a topic and path only based on smoothing-only-bin
11:    update new-topic, new-path
12:   else:
13:    sample -= smoothing-only-bin
14:   if new-topic == -1 and sample < topic-beta-bin:
15:    sample a topic and path only based on topic-beta-bin
16:    update new-topic, new-path
17:   else:
18:    sample -= topic-beta-bin
19:   if new-topic == -1:
20:    sample a topic and path only based on topic-term-bin
21:    update new-topic, new-path
```

---

---

**Algorithm 2** FAST SAMPLING BASED ON OVERESTIMATION

---

```
1: for word  $w$  in this document:
2:   get prior-smoothing-only-bin
3:   compute topic-beta-bin
4:   compute topic-term-bin
5:   norm = sum of the three bins
6:   sample = rand()
7:   sample * = norm
8:   new-topic = -1
9:   if sample < prior-smoothing-only-bin:
10:    compute the smoothing-only-bin
11:    update norm and sample
12:    if sample < smoothing-only-bin:
13:     sample a topic and path only based on smoothing-only-bin
14:     update new-topic, new-path
15:    else:
16:     sample -= smoothing-only-bin
17:   else:
18:    sample -= prior-smoothing-only-bin
19:   if new-topic == -1 and sample < topic-beta-bin:
20:    sample a topic and path only based on topic-beta-bin
21:    update new-topic, new-path
22:   else:
23:    sample -= topic-beta-bin
24:   if new-topic == -1:
25:    sample a topic and path only based on topic-term-bin
26:    update new-topic, new-path
```

---

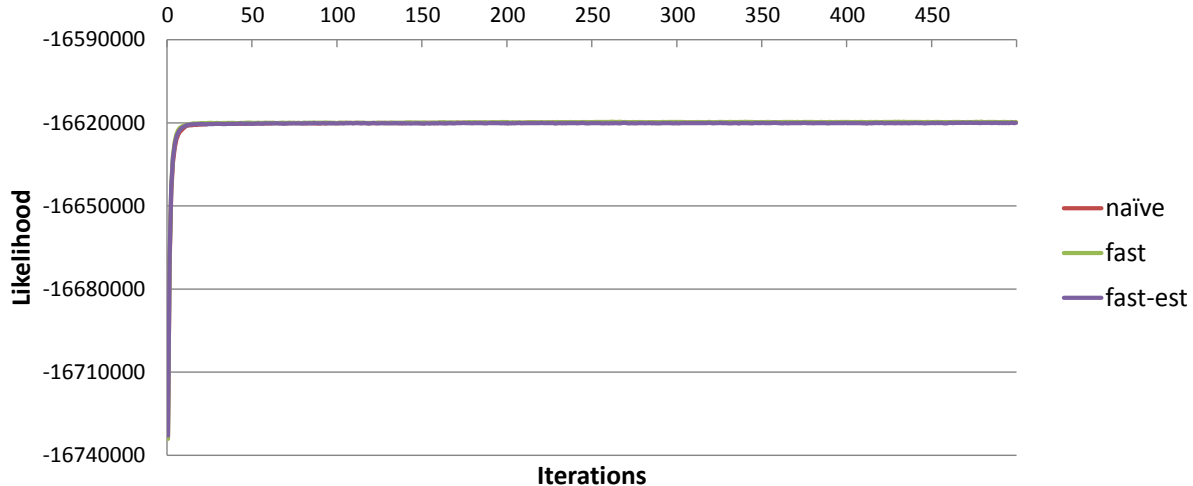


Figure 7: Likelihood comparison of three sampling methods: naive, fast and fast-est

### 4.3 Experiments

The experimental code is currently implemented in a standalone program which will be incorporated into the Mallet toolkit (as used in Section ??). We will compare the three methods, denoted as naive (using Equation 1), fast (using Equation 7) and fast-est (using Algorithm 1) on a bi-lingual corpus: 200 Chinese documents [?], 199 English documents [7], and the CEDICT providing word correlations [4].<sup>4</sup>

#### 4.3.1 Preprocessing

We extract the original vocabulary from all the Chinese and English documents, filtering out the punctuations, stopwords and less-frequent words. Then based on this original vocabulary, we try to find out the matched Chinese-English word pairs, and use these words in the matched pairs as the final vocabulary. In this experiment, 800 words are remained in the vocabulary. When load in the documents, we only consider the words contained in the vocabulary.

#### 4.3.2 Comparison

We ran the three methods for 500 iterations separately, and the topic number is set to be 20. Then we compared the three sampling methods by the likelihood, time per iteration and running time. From Figure 7, we can see that likelihood of the three methods is comparable, which is a sanity check of the correctness of the efficient sampling methods we proposed.

<sup>4</sup>We used these documents because they represent newswire, a common source of WISDOM data, and because they have gold-standard segmentations.

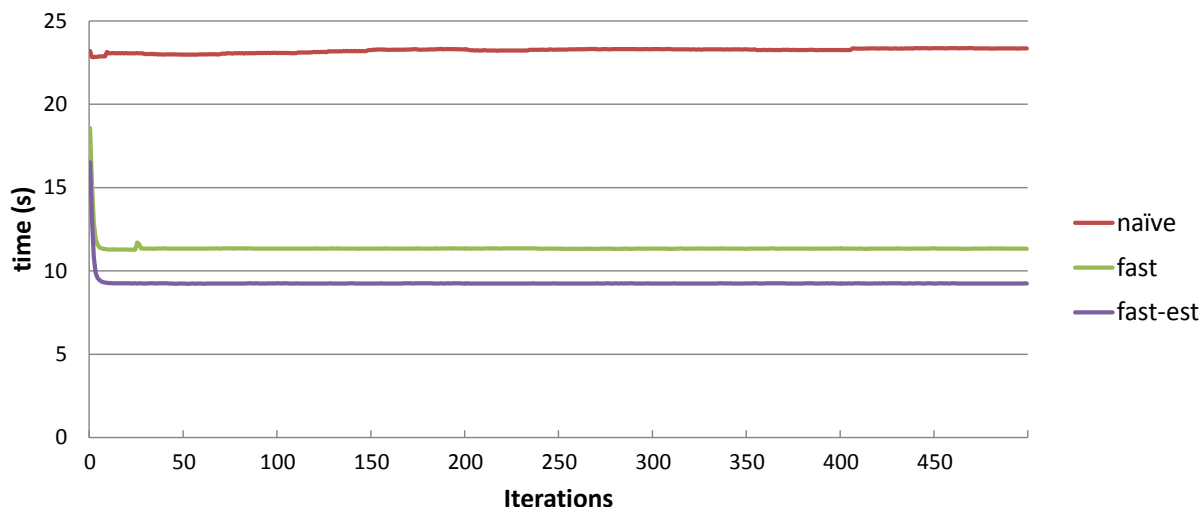


Figure 8: Time per iteration of three sampling methods: naive, fast and fast-est

Figure 8 shows the difference of the running time per iteration of the three methods. “fast” is almost twice as fast as “naive”, while “fast-est” is faster still. The final gain comes from just using the upper bound of the smoothing-bin instead of accurately computing it for most words. The time per iteration of naive sampling stays stable, while the two efficient sampling methods dropped at the very beginning and then reached a stable level; this is because the more complicated schemes benefit from the sparsity patterns inference favors. As a complementary figure, Figure 9 shows the cumulative the three methods, which clearly explained the time the efficient methods saved (e.g. what a user would experience).

Table 2 shows the detected topics by “fast-est”. Each topic usually has both Chinese words and English words. The English meaning is included in the brackets if it is a Chinese word.

#### 4.4 Integration with WISDOM

Fuller integration with WISDOM depends on the database representing tokenized terms and the language of a document. (While this could be integrated into the topic modeling code, it is more important for the words and document languages to be consistent with the rest of the WISDOM system.)

#### 4.5 Conclusion

We have developed an efficient way to do multi-lingual topic modeling more efficiently using a tree prior structure, and we have conducted experiments demonstrating the correctness of the approach and significant improvement of computation speed. This model can also be extended to incorporate correlations between words in the same language, a technique we have already begun to explore

Table 2: Twenty topics detected by the “fast-est” sampling method

Topic	Words
0	投资(investment), 增长(increase), 年(year), 出口(export), 继续(continue) 首(head chief first), 区(small region district), 前(ago earlier former previous), 商品(good), 明显(clear) 综合(composite), 体系(system), 形势(situation), 状况(situation state), 充分(full)
1	贸易(trade), 国家(country state nation), 中(among within), 市(city market), 次(second order) 位(place position), 增加(increase raise), 先进(advanced), 行业(industry business), 期间(time period) 海外(overseas), 情况(situation), 全面(total), 获得(receive get), advanced
2	上(last higher first previous), 说(say), 政府(government), 美国(us), government 形成(form), 提高(increase raise), 后(later back), 第一(first), 认为(feel believe think) 都(even already), 下(decline lower later second next), 汽车(car), 间(among time), health
3	国(country state nation), 地区(district region local), 产业(industrial industry), funds, country 资金(funds funding capital), well, good, since, 良好(fine good well) state, industry, 以来(since), get, 内(within)
4	国际(international), 达(clear), 利用(use), 成为(become), 总(head always general chief total) 项(term thing), 国有(public), 重要(major important significant), use, 关系(concern) 重点(key), 资产(assets), concern, public, become
5	business, 贷款(loan), trade, 商业(business trade), might, economic, investor, possible economy, administration, 投资者(investor), certain, move, management, loan, must
6	全(every), 产品(product goods), major, big, 水平(level), small, large, 支持(support back) every, product, back, 小(small young), around, support, huge, great
7	year, like, 加工(process working), 合同(contract), month, contract, 类(kind like type similar), 业务(business) construction, increase, work, 月(month), strong, job, raise, kind
8	银行(bank), 金融(banking financial finance), 期(term period time), bank, financial, banking, finance, 好(good well) term, 利润(profits), 7, profits, 运行(move run), 原(level cause former), well, 七(7)
9	提供(provide offer), offer, issue, people, rate, 保持(keep), six, close almost, end, 发行(issue), provide, 快(almost soon rate), bond, costs, 国外(overseas foreign)
10	世界(world), 保险(insurance), 会(group see meet), group, three, 今天(today), see, world 计划(program plan), 表示(show state say), 研究(study research), 影响(effect), largest, today, information, insurance
11	president, two, chief, earlier, 10, investment, ago, 2 二(2 two), 十(10), law, 能力(able ability), way, total, general, part
12	企业(firm company), 公司(firm company), 大(major huge big great large), 两(two), 三(3 three) 机构(agency), 五(5 five), 协议(agreement), city, 表明(known), amount 重大(great important major significant), 数字(number amount), known, 使用(use)
13	company, market, also, new, stock, share, program, last first, even, sales, recent, next, higher, sell, say
14	外(foreign), foreign, magazine, power, 势头(power situation), reports, 份(share magazine reports part newspaper) watch, 看好(support), 更(watch), situation, share, part, support, newspaper
15	工业(industry), 市场(market), 总额(total), 国内(domestic), 工作(construction job work), 集团(group) number, 问题(issue problem question), several, problem, 制度(system), 促进(boost) set, 规定(set provide rule), 条件(term), 倍(times)
16	新(new), 石油(oil), 改善(improve), office, 13, legislation, oil, agency, limit 法规(legislation), 销售(sell sales market), 办法(way), 办事处(agency office), 持续(continue), 十三(13), 范围(limit)
17	建立(construction), 加强(increase), 8, 外国(foreign), 政治(political), return, political, outstanding, 显著(outstanding) 加大(increase), 八(8), 最近(recently soon recent latest), 固定(set), 信贷(credit), 通用(common), 还(return)
18	经济(economic economy), 建设(construction), 还(return), 政策(policy), 基础(underlying), 经营(run) 管理(management administration), 人(person people man), 会议(conference meeting), 人民(people), 体制(system) 共(share general common total), 服务(service), 方式(way), meeting, 资本(capital)
19	已(already), 发展(development growth), 元(dollar first), 家(family home), 目前(currently), 高(high), make 主要(major principal), 使(use make cause), growth, already, take, high, 取得(get), another, family

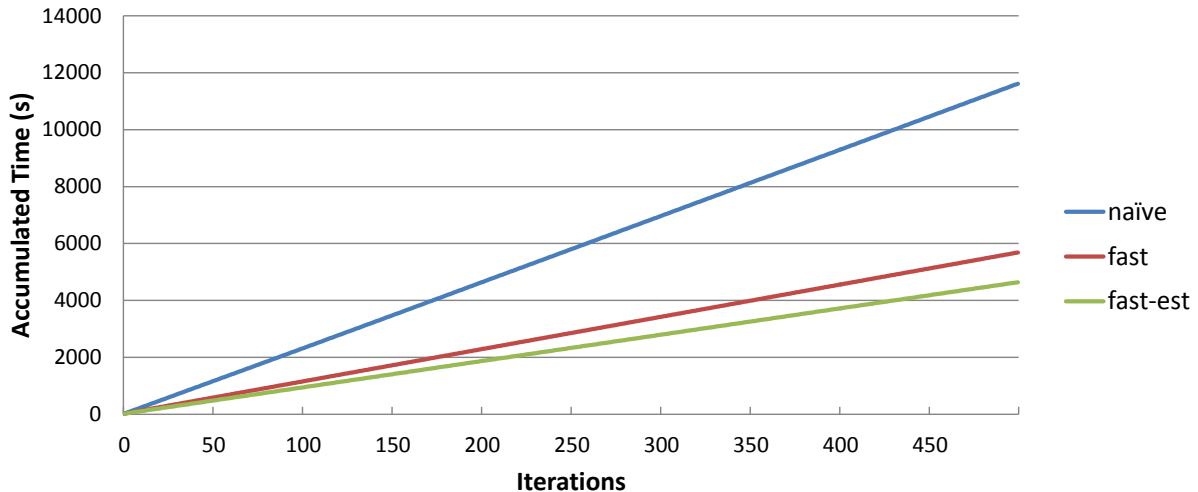


Figure 9: Accumulated running time of three sampling methods: naive, fast and fast-est

in [?]. We are currently working on further improving the implementation of our approach and incorporating it into Mallet, which has already been integrated with WISDOM (Section ??).

## References

- [1] David Andrzejewski, Xiaojin Zhu, and Mark Craven. Incorporating domain knowledge into topic modeling via Dirichlet forest priors. In *Proceedings of International Conference of Machine Learning*, 2009.
- [2] Jordan Boyd-Graber, David M. Blei, and Xiaojin Zhu. A topic model for word sense disambiguation. In *Proceedings of Empirical Methods in Natural Language Processing*, 2007.
- [3] Jordan Boyd-Graber and Philip Resnik. Holistic sentiment analysis across languages: Multilingual supervised latent Dirichlet allocation. In *Proceedings of Empirical Methods in Natural Language Processing*, 2010.
- [4] Paul Denisowski. CEDICT, 1997. <http://www.mdbg.net/chindict/>.
- [5] Yuening Hu, Jordan Boyd-Graber, and Brianna Satinoff. Interactive topic modeling. In *Association for Computational Linguistics*, 2011.
- [6] Claudia Kunze and Lothar Lemnitzer. Standardizing WordNets in a web-compliant format: The case of GermaNet. In *Workshop on Wordnets Structures and Standardisation*, 2002.
- [7] Mitchell P. Marcus, Beatrice Santorini, and Mary A. Marcinkiewicz. Building a large annotated corpus of English: The Penn treebank. *Computational Linguistics*, 19(2):313–330, 1993.

- [8] Radford M. Neal. Probabilistic inference using Markov chain Monte Carlo methods. Technical Report CRG-TR-93-1, University of Toronto, 1993.
- [9] Philip Resnik and Eric Hardisty. Gibbs sampling for the uninitiated. Technical report, University of Maryland, 2009. <http://www.umiacs.umd.edu/resnik/pubs/gibbs.pdf>.
- [10] Limin Yao, David Mimno, and Andrew McCallum. Efficient methods for topic model inference on streaming document collections. In *Knowledge Discovery and Data Mining*, 2009.