BBN/UMass Technical Term Generation  
Software Documentation 2015-06-01

Contents

[1 Overview 1](#_Toc420945349)

[2 Input Data Format 2](#_Toc420945350)

[3 Code Architecture and Structure 3](#_Toc420945351)

[3.1 Structure of Code/Scripts 3](#_Toc420945352)

[3.1.1 Environment setup file 3](#_Toc420945353)

[3.1.2 Source Code 3](#_Toc420945354)

[3.1.3 Scripts 4](#_Toc420945355)

[3.2 Software Versions Used 4](#_Toc420945356)

[3.3 Memory Requirements 4](#_Toc420945357)

[4 Generating and Evaluating Terms for a New Corpus 4](#_Toc420945358)

[4.1 Environment setup 5](#_Toc420945359)

[4.2 Setting the configuration variables 5](#_Toc420945360)

[4.3 Embedding Model Creation 6](#_Toc420945361)

[4.3.1 Generate phrases 6](#_Toc420945362)

[4.3.2 Generate the embedding model 7](#_Toc420945363)

[4.4 Discipline-level data generation 8](#_Toc420945364)

[4.5 Term Generation 9](#_Toc420945365)

[4.5.1 Adding terms to pos.list 9](#_Toc420945366)

[4.5.2 Adding terms to neg.list 10](#_Toc420945367)

[4.5.3 Training the svm classifier 10](#_Toc420945368)

[4.6 Evaluation 11](#_Toc420945369)

[4.6.1 Entropy-based evaluation 11](#_Toc420945370)

[4.6.2 Document Coverage metric 12](#_Toc420945371)

[4.7 Cleaning script 12](#_Toc420945372)

[5 Generating Terms for a New Discipline in an Existing Corpus 12](#_Toc420945373)

# Overview

This document describes the steps to generate technical terms, given an input corpus. This process involves creating a word-embedding model from the full text of all documents in the corpus. This model uses the co-occurrence patterns in the corpus to assign each word a location within a multi-dimensional semantic space.

Given this model, one can then generate a list of terms for a given discipline (subset of the corpus) by collecting some specific information about that discipline and then manually annotating a few (c. 100) of the terms as either good technical terms or not useful terms. These annotations are then used to train a (SVM – Support Vector Machine) classifier, which uses the word-embedding information about each term to select terms from the full set that are likely to be good technical terms. One can also measure the entropy of the contexts in which terms occur, where low entropy scores indicate terms that tend to occur in specific types of contexts, which is often true of useful technical terms.

# Input Data Format

Term generation is based on a large corpus of text documents. In our FUSE work, we have used data from the following four corpora:

* English patents (LNUS)
* English Web of Science (WOS)
* Chinese patents (LNCN)
* CNKI (China National Knowledge Infrastructure)

The term generation software uses a standard document-per-line format for the input data. (For FUSE, we derived this format from the individual XML files in the source.)

Each line of an input data format document is space-separated and has the following fields:

<doc id> <year> <text>

To use this component with other corpora, input files should be derived that follow this data format convention. The <text> part of the data should be processed so as to make the words surrounding each word easily recognizable, i.e. tokenized, punctuation removed, and preferably lowercased. This applies to both English and Chinese data files.

# Code Architecture and Structure

## Structure of Code/Scripts

The software is available on FUSEnet at the location given below:

/nfs/fuse-scilinus-01/u10/term\_generation/software

This package contains the following components:

### Environment setup file

setup.sh file creates the directories to which the output and log files will be written while running the code.

### Source Code

* src/factorie-factorie\_2.11-1.1

Contains factorie release 1.1 which contains the embedding model code with small modifications.

The code has been pre-compiled and the jar is placed in target/factorie\_2.10-1.1-SNAPSHOT-nlp-jar-with-dependencies.jar.

If the code is modified, run the following command in the factorie-factorie\_2.11-1.1 directory to package the code and generate the modified jar :

mvn -Dmaven.test.skip=true package -Pnlp-jar-with-dependencies

* src/technicalterms

Contains code to generate phrases from the corpus as part of the preprocessing step and code to evaluate terms based on entropy and doc coverage.

The code has been pre-compiled and the jar is placed in

target/technicalterms-1.0.jar.

If the code is modified, run the following command in the technicalterms directory to package the code and generate the modified jar:

mvn clean package

* src/python

Contains code to train svm classifier to identify technical terms for discipline level terms and generating discipline level data.

### Scripts

The scripts/ directory contains wrapper scripts to create embedding model, generate discipline level-data and classify terms as technical/non-technical.

## Software Versions Used

* python 2.6 - This has be installed on the server.
* scala 2.10.4 – This is automatically downloaded via maven and included in the dependencies while building the jar. The delivered jar file already contains this version of scala and does not need to be installed separately.
* Maven version>3 – This has to be installed on the server if the source code needs to be recompiled.

## Memory Requirements

All the scripts provided have an input variable to set the memory.

Currently it is initialized to 3G.

While building the embedding models and entropy calculation, we found that the memory scales with respect to the size of input data.

For these processes, it is recommended to start with 50G of memory and increase it as required.

The maximum memory requirement that we had was for CNKI A78 discipline entropy evaluation which required ~150G of memory.

# Generating and Evaluating Terms for a New Corpus

This section describes the process for generating terms from an input corpus.

The instructions for the executing the code mentions “qsub” calls and assumes that the code is being executed on a cluster managed by an Oracle Grid Engine. This is helpful to manage memory and resource allocation. But even if that is not the case, the scripts can still be executed using “sh <scriptname>” command assuming that enough memory is available for executing the process.

## Environment setup

Begin by copying the /software folder to the directory area where you plan to run the system. (Note that the scripts in /scripts need to be run from that directory, since they use relative paths.)

Execute the following script from the /software folder

sh setup.sh

This generates the following directories.

* output/model
* output/terms
* output/eval
* logs

## Setting the configuration variables

Modify the following variables in the scripts/config file

- TECHTERMS\_DIR: This gives the absolute path where the software is copied.

Currently in the FUSEnet server, this is set to

/nfs/fuse-scilinus-01/u10/term\_generation/software

- DATA\_FILE: This gives the absolute path where the corpus file is stored. The contents of this file should be of same format mentioned under the Data section.

- DISP: This gives the name of the discipline from which technical terms have to be generated.

For eg : a41,a42 etc..

- DISP\_MAPPING\_FILE: This gives the path of the discipline-level-mapping file, which indicates which documents in the input data file, belongs to this discipline.

## Embedding Model Creation

This section describes the steps required to identify valid phrases from the existing dataset and introduce them into the data (for eg: if we identify “new york” as a phrase, this is replaced by new\_york so that this can be created as a single term instead of two). An embedding model is built using this data with phrases as input.

### Generate phrases

From the scripts/ directory, execute the following script to generate data with phrases up to 6-grams.

qsub ./gen\_phrases.sh

Check the following log files to see if the scripts have executed successfully:

* logs/gen\_phrases.out
* logs/gen\_phrases.err

The following output files are generated:

* output/model/<data\_file\_name>.2gram.data
* output/model/<data\_file\_name>.2gram.phrase
* output/model/<data\_file\_name>.3gram.data
* output/model/<data\_file\_name>.3gram.phrase
* output/model/<data\_file\_name>.4gram.data
* output/model/<data\_file\_name>.4gram.phrase
* output/model/<data\_file\_name>.5gram.data
* output/model/<data\_file\_name>.5gram.phrase
* output/model/<data\_file\_name>.6gram.data
* output/model/<data\_file\_name>.6gram.phrase

### Generate the embedding model

From the scripts/ directory, execute the following script to generate embedding model.

qsub ./gen\_embeddings.sh

The following parameters have been set in the gen\_embeddings.sh. This can be modified if required.

* threads : Number of threads available to run the process
* min\_count : Minimum number of times a term has to occur in a corpus. If the count of a term is less than min-count, embeddings are not learned for that term. Default value is 10.
* ignore-stopwords : If set to true, stopwords are ignored and embeddings are not learned for them else they are included in the vocabulary. Default value is true.

In order to check for a stopword, the term is compared against the list given in factorie package :

src/factorie-factorie\_2.11-1.1/src/main/scala/cc/factorie/app/nlp/lexicon/StopWords.scala

* max-vocab-size : Maximum number of words in a corpus for which embeddings are learned. Default value is 2,000,000.
* sample : sampling rate to reduce the number of gradient updates on high frequency words. Default value is 0.00001

Check the following log files to see if the scripts have executed successfully:

* logs/gen\_embeddings.out
* logs/gen\_embeddings.err

The following output files are generated:

* output/model/<data\_file\_name>.6gram.embeddings
* output/model/<data\_file\_name>.6gram.vocab

The <data\_file\_name>.vocab contains the terms from the entire corpus. Each line of the file is space separated and has the following format:

<term> <term\_frequency>

The output/model/<data\_file\_name>.embeddings contains the vector representation of the terms. Each line of the file is space separated and has the following format:

<term> <dim1> <dim2> …… <dimn>

where dimx refers to the xth dimension of the vector.

The default value for n is 200.

The term can be a unigram,bigram or higher n-grams. N-grams are underscore separated.

## Discipline-level data generation

This section describes the steps to generate discipline-specific data from the input data file. We perform this step to limit the vocabulary of terms to the discipline of interest instead of performing the downstream steps on the entire vocabulary. This step also makes the annotation easier. For e.g.: Discipline “Computer Science” would have a very different set of technical terms as opposed to “Material Science”.

From the scripts/ directory, execute the following script,

qsub ./gen\_disp\_data.sh

This script uses the input discipline-mapping file given in the config file against the option “DISP\_MAPPING\_FILE” and matches the id with those in the input data file (which is the “DATA\_FILE” option in config). To maintain consistency, we assume that the mapping file is tab-separated in the following format:

<doc-id> <year>

where the doc-id is consistent with the doc-id(i.e. the first column) in the data file.

Check the following log files to see if the scripts have executed successfully:

* logs/gen\_disp\_data.out
* logs/gen\_disp\_data.err

The following output files are generated:

* output/model/<data\_file\_name>.<discipline\_name>.6gram.vocab
* output/model/<data\_file\_name>.<discipline\_name>.6gram.data

This script also creates a sub-directory with discipline name under output/terms/<discipline\_name>

## Term Generation

This involves training a semi-supervised svm classifier to identify the technical terms using embedding of the terms as input. This method requires a small amount of manual annotation.

Under output/terms/<discipline\_name> folder, two new files - pos.list and neg.list have to be created. pos.list should contain terms which are considered to be technical ie positive examples and neg.list should contain terms which would not be considered technical. The terms generated from the discipline level data in Step 4(.vocab file) above has to be used to populate these lists.

### Adding terms to pos.list

A list of 50-60 technical terms from the discipline-specific vocabulary are identified and added to this list, one term per line.

Please note that the n-grams have to be underscore separated, i.e. in the same format as they appear in the vocab files.

We used the following methods to identify the positive list.

* Use terms from discipline-specific glossary. This involves scanning the vocab files and identifying the terms which matches terms in the glossary.
* Picking up the terms from the glossary might result in a small subset of related terms based on the bias of the annotator. In order to circumvent this issue, execute the following script under scripts/ folder to generate cluster of terms:

qsub ./kmeans.sh

In kmeans.sh file, currently the number of clusters to run kmeans is set to 400. If required this can be modified by re-setting the input variable K.

Picking a few terms from each of the relevant clusters would ensure higher coverage.

Check the following log files to see if the scripts have executed successfully:

* logs/kmeans.out
* logs/kmeans.err

The output clusters generated will be present in the following file:

output/terms/<discipline\_name>/kmeans.out

### Adding terms to neg.list

The top 60-70 most frequent terms from the discipline-specific vocabulary file can be used as negative examples. They also have to be added one per line and in the same format as pos.list.

### Training the svm classifier

From the scripts/ folder execute the following script:

qsub ./gen\_terms.sh

Input parameter : prob\_cutoff .

This is currently set to 0.8. All terms, which are assigned a probability value above this, is considered to be a technical term.

The following log files are generated:

* logs/gen\_terms.out
* logs/gen\_terms.err

Output files generated:

* output/terms/<discipline\_name>/pos\_pred.list – **contains terms classified as technical**
* output/terms/<discipline\_name>/neg\_pred.list – **contains terms classified as non-technical**
* output/terms/<discipline\_name>/pos\_pred\_prob.list – contains terms classified as technical with probability value
* output/terms/<discipline\_name>/pos\_pred\_prob.list – contains terms classified as non-technical with probability value

If the qualities of terms have to be improved, then perform the steps above again with a new list of positive (pos.list) and negative (neg.list) examples. You can also remove a subset of examples from the lists or modify and rerun the classifier.

## Evaluation

### Entropy-based evaluation

This generates entropy values for each term in the predicted positive list.

From the scripts/ folder, execute the following.

qsub ./gen\_entropy.sh

The following input parameters have been set in the file. They can be modified if required:

* alpha : This is Laplace Smoothing parameter. Currently it is initialized to 0.01.
* NUM\_TERMS: This indicates the number of top predicted positive terms, which have to be evaluated. The default value has been set to 5000. This can be increased up to the size of the total predicted positive list if required.

The following output file is generated:

output/eval/<discipline\_name>/entropy.<NUM\_TERMS>.list

This file contains the entropy values of top predicted positive

The following log files are generated:

logs/gen\_entropy.out

logs/gen\_entropy.err

Each line in the file is tab-separated and contains a term and its corresponding entropy value. The terms are sorted in increasing order of their entropy values i.e. the lower entropy terms are listed at the top of the file and higher entropy terms at the bottom.

In case of english data, the code sifts out terms which are stopwords and these would not appear in the output file.

### Document Coverage metric

This generates the percentage of discipline level documents, which contains at least one term in the list.

From the scripts/ folder, execute the following

qsub ./gen\_doccoverage.sh

The following output file is generated.

output/eval/<discipline\_name>/doccoverage.txt

The following log files are generated.

logs/gen\_doccoverage.out

logs/gen\_doccoverage.err

## Cleaning script

The cleanup script removes log files from the /log directory. This should be executed periodically if the logs consume too much disk space.

From the scripts/ directory, execute the following to cleanup all the log files.

sh clean\_logs.sh

# Generating Terms for a New Discipline in an Existing Corpus

If the embedding model for an existing corpus has been created earlier and only terms for a new discipline need to generated, then modify the config file entries pertaining to DISP and DISP\_MAPPING\_FILE and execute Steps 4-6 given under the previous section.