LEAN: Lucene Tools for Text Analytics

Prepared by:

Richard Boyd
Senior Research Scientist
richard.boyd@gtri.gatech.edu

Ashley Scripka Beavers
Research Scientist I
ashley.beavers@gtri.gatech.edu

Information and Communications Laboratory
Georgia Tech Research Institute
250 14th St. NW
Atlanta, GA 30318

8 October 2015

Table of Contents

I. Introduction

LEAN is a set of Java tools for generating term-frequency matrices from text documents. The LEAN tools are designed for compatibility with the GTRI/GA Tech *SmallK* software distribution, which consumes term-frequency matrices and performs hierarchical and flat clustering.

The LEAN software distribution currently consists of two tools: *DocIndexer* and *LuceneToMtx*. The DocIndexer application ingests documents in various formats and encodings, analyzes them with a user-configurable Lucene analyzer, and generates a Lucene inverted index. The LuceneToMtx application reads the index, performs optional filtering on the terms, and generates a term-frequency matrix with matching dictionary and document files.

The essential component in the LEAN toolset is Lucene[1]. Lucene is a state-of-the art, open-source text indexing and search library that performs tokenization and filtering on a stream of text. Surviving tokens are written to an inverted index. Lucene includes a large set of text analyzers and filters for multiple languages, which incorporate many advanced natural language processing (NLP) techniques. It also provides an extension mechanism for developing custom analyzers and filters, which LEAN takes full advantage of.

The LEAN tools were developed for the purpose of cleaning up text for document clustering. Cluster quality is significantly impacted by the quality of the input text; the garbage-in garbage-out principle definitely applies. For *formal* documents such as news articles and books, the NLP tools incorporated into Lucene's analyzers perform well. For *informal* text, such as Twitter, web chat, and SMS messages, Lucene's analyzers tend to perform poorly. The standard corpora for formal text, on which many NLP tools are trained, simply do not exhibit the wide variety of self-expression seen in informal text. Additionally, these formal analyzers poorly handle more recent innovations such as URLs, emoticons, and other aspects of computer-mediated communication.

Therefore we have taken advantage of Lucene's extension mechanism to develop an improved set of tools for informal text. We use Twitter as our representative source of informal text. Twitter is extremely popular, the data is readily available, and it has been the subject of much study by NLP and text analytics researchers. Twitter has also demonstrated its importance for real-time documentation of world events, such as in the Egyptian revolution of 2011.

The DocIndexer application includes a custom Lucene analyzer for Twitter. This analyzer incorporates a Twitter-aware tokenizer and part-of-speech (POS) tagger from the TweetNLP[2] toolset developed at Carnegie Mellon University. The analyzer uses the POS tag to help it distinguish between tokens relevant to the meaning of the tweet (such as proper nouns) and irrelevant tokens (such as emoji). Although the Tweet NLP tools are extremely capable, they simply cannot identify every nonstandard construct appearing in a tweet. Thus the LEAN Twitter analyzer incorporates additional custom token filters that correct issues identified via an empirical study of thousands of tweets.

At present, the LEAN tools run as a single process; both Mac OSX and Linux are supported. Scalability was not the goal of this initial release of the toolset. It is possible to integrate our Lucene analyzers into Solr[3] and develop a distributed, fault-tolerant system capable of scaling to large numbers of documents in multiple languages, if the needs of the application require such performance.

[1] lucene.apache.org
[2] http://www.ark.cs.cmu.edu/TweetNLP/
[3] http://lucene.apache.org/solr/

II. Installation Instructions

II.1 Source Code

The LEAN source code is hosted in a public git repository at www.github.com/smallk/lean.

II.2 Java 8 Standard Edition and NetBeans IDE for Java Development

The LEAN tools were developed with the Java 8 JDK. They can be built with either the NetBeans IDE or with Apache Ant.

Thus the first step in the installation process is to install Java 8 SE[4] (Standard Edition) on your system.

Optional: If you intend to use the NetBeans IDE for Java Development to edit this code, you can install both NetBeans and Java 8 JDK[5] as a bundle. Download the distribution appropriate for your platform and follow the installation instructions[6].

If you wish to install just Java 8 SE, download the appropriate distribution here[7] and follow the installation instructions[8].

For Linux, system-wide installation is probably easiest, since most Linux package managers have installation packages for the JDK. Alternatively, the JDK tarball can be unzipped to any folder and the path to JAVA_HOME set as indicated in the next section.

II.2.1 Mac OXS Startup Script

Add the following to your ~/.bash_profile script (or whatever startup script you use):

```
export JAVA_HOME=$(/usr/libexec/java_home)
```

11.2.2 Linux Startup Script

Add the following to your ~/.bashrc script (or whatever startup script you use):

```
export JAVA_HOME=/path/to/jdk1.8.x_x/jre/
```

To access the new java binary, you can create a symlink in /usr/local/bin:

```
ln -s /path/to/jdk1.8.x_x/jre/bin/java /usr/local/bin/java
```

II.3 Apache Ant

Apache Ant[9] is a Java library and command-line tool primarily used to build Java applications.

[4] http://www.oracle.com/technetwork/java/javase/downloads/index-jsp-138363.html
[5] http://www.oracle.com/technetwork/java/javase/downloads/jdk-netbeans-jsp-142931.html
[6] http://www.oracle.com/technetwork/java/javase/downloads/install-jdk6-22nb691-177131.html
[7] http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
[8] http://docs.oracle.com/javase/8/docs/technotes/guides/install/install_overview.html
[9] http://ant.apache.org/

II.3.1 Install on Mac OSX

On MacOSX we recommend using Homebrew[10] as the package manager. Homebrew does not require sudo privileges for package installation, unlike other package managers such as MacPorts. Thus the chances of corrupting vital system files are greatly reduced with Homebrew.

If you use Homebrew, ensure that your PATH is configured to search Homebrew's installation directory first. Homebrew's default installation location is `/usr/local/bin`, so that location needs to be first on your path. Check this by running this command from a terminal window:

```
cat /etc/paths
```

If the first entry is not `/usr/local/bin`, you will need to edit this file. Since this is a system file, it would be wise to create a backup. If permission is denied to modify this file, use `sudo` in front of the commands. Move the line `/usr/local/bin` so that it is on the first line of the file. Save the file, then close the terminal session and start a new terminal session so that the path changes will take effect.

After starting your new terminal session, make sure your homebrew packages are up-to-date by running the following commands:

```
brew update
brew upgrade
brew cleanup
brew doctor
```

To install Apache Ant via Homebrew:

```
brew install ant
```

II.3.2 Install on Linux

We recommend using a Linux package manager for installation and configuration of Apache Ant. Ant is available in most Linux distributions' repositories. A simple example of installation would be:

```
apt-get install ant
```

II.3.3 Check installation

Ensure that you have an appropriate version of Apache Ant (1.9.x) by running:

```
ant -version
```

You should see something similar to the following:

```
Apache Ant(TM) version 1.9.4 compiled on April 29 2014
```

If you do not see the expected version number, you may have an older version of ant installed on your system, or an ant config file may be interfering with the new installation. Try the command again, this time with a fully-qualified path to the ant binary, and also specifying the `--noconfig` option:

```
path/to/apache-ant-folder/bin/ant --noconfig -version
```

[10] http://brew.sh/

III. Build Instructions

The LEAN tools can be built within the NetBeans IDE for Java Development or on the command-line via Apache Ant. Either method will build the application's jar file and place it in the `dist/` subdirectory of the application folder.

III.1 Build using NetBeans IDE

Launch the NetBeans IDE and choose 'Open Project' from the File menu. Browse to the LEAN repository on your system and select the 'DocIndexer' project. Click the 'Open Project' button to let the project load.

Build the project by going to the 'Run' menu and selecting 'Clean and Build Project.' In the 'Output' panel, you should see 'BUILD SUCCESSFUL' in green text.

Repeat the build process for LuceneToMtx. Both projects should build with no errors. For convenience, you can easily use NetBeans to make modifications to and customizations of the code and rebuild the projects.

III.2 Build using Ant

On the command-line, navigate to the DocIndexer subdirectory in the LEAN repository on your system. Run

```
ant jar
```

You will see some output, ending with 'BUILD SUCCESSFUL' followed by a time measurement. If there are errors, you will see 'BUILD FAILED' along with information about the errors.

Navigate to the LuceneToMtx subdirectory in the LEAN repository and repeat the build command.

If you have trouble with this step, try using a fully-qualified path to the ant binary, along with the `--noconfig` option as indicated above.

IV. Usage of DocIndexer

IV.1 Overview

The DocIndexer application reads in documents, extracts the text content, analyzes it with a predefined or custom Lucene analyzer, and writes the resulting tokens to a Lucene index. The LuceneToMtx application reads this index and generates a term-frequency matrix from it.

There are two modes of operation for DocIndexer. In *analysis* mode, DocIndexer displays the tokenized and analyzed text without writing anything to a Lucene index. This mode of operation is useful for investigating and experimenting with various analysis techniques. In *normal* mode, DocIndexer fully analyzes the documents and writes the tokens to a Lucene index.

DocIndexer uses Apache Tika[11] to perform content extraction from documents. Tika is a Java library that exists for the sole purpose of extracting content and metadata from a wide variety of file formats. Tika can parse text files in various encodings, PDF files, web pages, Microsoft Word files, and many, many more.

DocIndexer's normal mode of operation is to open a directory on disk and recurse through the file tree rooted at that folder, extracting content from all recognized document formats. DocIndexer prints a message to the screen if it encounters a file it cannot recognize.

DocIndexer can also ingest raw Twitter JSON data, either from files on disk or from a MongoDB database. Twitter's servers return data in a known format (UTF8-encoded JSON), so TIKA is not needed to parse it (the data files must have a .json extension). We have written our own fast Twitter data parser that simply pulls the tweet from JSON files as fast as it can. For Twitter data stored in a Mongo database, DocIndexer uses the MongoDB Java Driver.

IV.2 Usage

The DocIndexer application is an executable jar file. If you run the jar file without any command-line arguments, or with the `-h` or `--help` options, the application will output the command-line options supported. For instance, from the top-level LEAN folder, display the DocIndexer's command-line options by running this command:

```
java -jar DocIndexer/dist/DocIndexer.jar
```

You should see the following output:

```
Usage: java -jar /path/to/DocIndexer.jar
            --config    <filename>   YAML configuration file
           [--analyze        n   ]   show analysis for first n documents; do not index
```

Only the first parameter, `--config`, is required. The meanings of the parameters are as follows:

1. `--config`: Path to the YAML configuration file which contains inputs for running the DocIndexer application. A detailed description of the contents of this configuration file is provided in the following section.

[11] http://tika.apache.org/

2. `--analyze`: A non-negative integer with a default value of 0. Nonzero values place the application in analysis mode, in which it displays analysis results for the first n documents only. These documents will not be indexed. Analysis mode is used for investigating the effects of different Lucene analyzers on the text. If this parameter is omitted, the application runs in normal mode, and it will analyze all of the input documents and generate a Lucene index.

DocIndexer prints an updated document count after every 10th document or 1000th tweet.

To illustrate the use of DocIndexer, the following command displays the analysis results from the first 5 documents (tweets, in this case) in the 'samples.json' file provided in the `data/twitter_test/` folder of the LEAN repository. Run the command from the top-level LEAN folder, assuming it contains a suitably-prepared YAML configuration file called `my_config.yml` (to be described in the next section). The output will look similar to this, with paths appropriate for your system:

```
java -jar DocIndexer/dist/DocIndexer.jar --config my_config.yml --analyze 5

DocIndexer version 0.2.

Configuration data:

         Configuration file: /path/to/config.yml
         Default slang file: /path/to/config/default_slang.txt
      Default stopword file: /path/to/config/default_stopwords.txt
         User stopword file: <NONE>
      Default spelling file: /path/to/config/default_spellings.txt
         User spelling file: <NONE>
                  Input dir: /path/to/data/twitter_test
                     outdir: /path/to/index
                   analyzer: CUSTOM_TWITTER
     Disable custom filters: No
                 Ignore URLs: No
            Ignore hashtags: No
         Ignore at-mentions: No
            Ignore retweets: No
            Ignore numerals: No
           Disable stemming: No

Java runtime max memory: 3641 MB.

Analysis results for file /path/todata/twitter_test/samples.json, 5 tweets max:

[1]     \u201cThe more toppings a man has on his pizza, I believe, the more manly he is,\u201d
Cain opined.  #OWS http://t.co/iPsUdnGw
start:    10 end:    14   length:     4      tag: A      token: more
start:    15 end:    23   length:     7      tag: N      token: topping
start:    26 end:    29   length:     3      tag: N      token: man
start:    41 end:    46   length:     5      tag: N      token: pizza
start:    50 end:    57   length:     7      tag: V      token: believe
start:    63 end:    67   length:     4      tag: A      token: more
start:    68 end:    73   length:     5      tag: A      token: manly
start:    87 end:    91   length:     4      tag: ^      token: cain
start:    92 end:    98   length:     5      tag: ^      token: opine
start:   101 end:   105   length:     4      tag: #      token: #ows
start:   106 end:   126   length:    20      tag: U      token: http://t.co/ipsudngw

        <remaining output omitted>
```

The output begins with a listing of the options loaded from the config file (displayed in this example with generic paths). The amount of memory available to the Java Virtual Machine is displayed next. Following this is a statement about the file being analyzed. After this you can see the first document (tweet text) and the tokens that emerged from the analysis and filtering operations. The 'start' and 'end' values are the character offsets from the start of the tweet text for each token (position numbering begins with 0). The tags for formal documents are determined by Lucene's standard tokenizer. The tags for Twitter documents are described in the annotation guidelines for the CMU

Twitter part-of-speech tagger[12], which can be found in the `docs/` folder of the LEAN repository (see the document called '`TweetNLP.pdf`').

Observe that the TwitterAnalyzer is able to remove the explicit Unicode codepoints `\u201c` and `\u201d` (which represent double quote characters); remove stopwords and punctuation; convert the text to lowercase; identify the Twitter hashtag '#ows' (Occupy Wall Street); and preserve the URL at the end of the tweet. Many other operations are possible with this analyzer.

To generate a Lucene index from the documents, run the command without the `--analyze` option. This will generate the Lucene index and write it to the location specified in the config file. All tweets in the input file will be analyzed. The Twitter test file only contains a small number of tweets, so we do not recommend trying to generate a term-frequency matrix from the index.

Occasionally DocIndexer may generate a 'TikaException' – this means that the Tika library was unable to extract content from the document, likely due to the format not being supported.

Note: Depending on the number of documents being processed, you may find it necessary to increase the amount of memory allocated to the JVM. Normally one does this by specifying the following two parameters on the command line when running the application.

```
-Xms<size>     specifies the initial Java heap size (default is 2Mb)
-Xmx<size>     specifies the maximum Java heap size (default is 64Mb)
```

For example, to set the minimum at 64Mb and the maximum at 256Mb:

```
java -Xms64m -Xmx256m -jar DocIndexer/dist/DocIndexer.jar …
```

DocIndexer prints out the maximum available memory when it launches. You should see this value update with a new value close to that specified in the script.

However, since DocIndexer is built and bundled into an executable jar file, this method may not work if you run this command directly in your terminal. If it does not work on your system, copy this command line into a shell script, then execute the shell script.

For instance, to use the shell script method, create a new file called run.sh, copy the command line above into it, and then execute the script with this command:

```
sh run.sh
```

You should then see DocIndexer report a max memory value similar to what you specify.

The Lucene index is written as the documents are processed. If you should happen to get an out-of-memory error, the documents indexed prior to that occurrence will still be written to disk. The actual Lucene index consists of multiple files. To completely delete a Lucene index, simply delete everything in the `index/` folder with this command:

```
rm index/*
```

IV.3 YAML Configuration File

DocIndexer requires the use of a YAML configuration file to set various runtime options. Three default configuration files are included in the LEAN repository and include detailed inline

[12] http://www.ark.cs.cmu.edu/TweetNLP/

comments. Use these as the starting point for your own configuration files by copying them, renaming them, and setting the paths as appropriate for your system. These default files are: '`default_docs_config.yml`', which configures the DocIndexer to recurse through a directory tree of documents; '`default_twitter_config.yml`', for processing Twitter JSON data from a directory tree; and '`default_mongo_config.yml`', for processing Twitter data from a MongoDB database.

We strongly recommend that users enter absolute paths for the config options that require the location of a directory or a file.

The config files provide a mechanism for users to contribute their own stopwords and spelling/slang correction mappings. These are specified via the 'user_stopword_file' and the 'user_spelling_file' entries. The format of these files is discussed next.

IV.3.1 Stopword File Format

DocIndexer can use up to two files containing stopwords. One file contains a default set of stopwords, is loaded at startup, must be named '`default_stopwords.txt`', and is stored in `config/default_stopwords.txt`. The other file is optional and can be provided by the user, with the path to the user stopword file specified in the config file.

The format of the stopword file is simply one stopword per line, i.e.:

```
a
am
an
and
...
```

The user stopword file can be used to remove specific unwanted tokens. Users can have different user stopword files for different data sets. Entries in the stopword files should be in lowercase, since the custom analyzers remove stopwords after converting the text to lowercase.

IV.3.2 Spelling File Format

DocIndexer can use up to two files containing spelling correction mappings. One file contains the default list of common spelling errors, is loaded at startup, must be named `default_spellings.txt`, and is stored in `config/default_spellings.txt`. The other file is optional and can be provided by the user, with the path to the user spelling file specified in the config file.

The format of the spelling file is as follows: each line consists of a correctly spelled word, followed by whitespace, then a comma-separated list of one or more misspellings. For instance:

```
absence               absense, absance
dependent             dependant
pronunciation         pronounciation
repetition            repitition
restaurant            restarant, restaraunt
…
```

Here you can see the correct spelling on the left and common misspellings on the right. The default spelling file contains unambiguous misspellings of common words (misspellings are extremely common in informal text).

The custom analyzers apply spelling correction near the end of the processing chain, just prior to stemming. The user spelling file can be used to correct common domain-specific misspellings, or simply to augment the existing default list with additional words or abbreviations.

As mentioned above, the paths to stopword files and spelling files are specified in the config file. The next section discusses the contents of the config files.

IV.3.3 The Default Config File for General Documents

To process a set of documents stored in a filesystem hierarchy, start with the config file called '`default_docs_config.yml`'. This file is stored in the top-level LEAN folder. Make a copy of this file, rename it, and then set the paths appropriately for your system. Here is a listing of the contents of this file, followed by a discussion of the various items:

File 'default_docs_config.yml':

```
##############################################################################
#                                                                            #
#                       GENERAL CONFIGURATION DETAILS                        #
#                                                                            #
#     Configure DocIndexer to processes documents in a directory tree.       #
#                                                                            #
##############################################################################

# [REQUIRED] Specify the full path to the LEAN repository's 'config' folder.
# This folder contains default stopword and spelling files, among other items.
# Items in this folder are loaded automatically at startup.

config_path: /path/to/lean/config

# [REQUIRED] Write the Lucene index to this folder.  If this folder does not
# exist it will be created.

outdir: /path/to/index

# [REQUIRED] Specify the analyzer to use.  An analyzer consists of a tokenizer
# and a chain of zero or more token filters.  The filters perform various
# transformations on the tokens as they pass down the chain.  The first four
# analyzers are provided by Lucene and are not customizable; the 'custom'
# analyzers can be easily altered and recompiled.

# org.apache.lucene.analysis.core.WhitespaceAnalyzer      Split text on whitespace only
# org.apache.lucene.analysis.standard.StandardAnalyzer    Lucene's default text analyzer
# org.apache.lucene.analysis.standard.ClassicAnalyzer     Lucene's StandardAnalyzer pre v3.1
# org.apache.lucene.analysis.en.EnglishAnalyzer           Lucene's English-specific text analyzer
# analyzers.FormalAnalyzer                                GTRI analyzer for formal documents
# twitter.TwitterAnalyzer                                 GTRI custom analyzer for Twitter data

analyzer: analyzers.FormalAnalyzer

# [REQUIRED] Specify the full path to the root folder of the file tree to be
# analyzed.  DocIndexer will begin at this root folder and recursively process
# all documents in supported formats throughout the tree.

indir: /path/to/input_folder_root

# [OPTIONAL] Specify the absolute path to a 'user' stopword file.  Use this
# file to give DocIndexer additional stopwords that should be removed from the
# token stream, but that are not contained in the default stopword list
# (found in the config folder).  This file can also be used to remove specific
# tokens that may not be of interest for a given data set.  The file name does
# not have to be 'user_stopwords.txt'.  To NOT use a user stopword file,
# comment the following line.

#user_stopword_file: /path/to/user_stopwords.txt

# [OPTIONAL] Specify the absolute path to a 'user' spelling file.  Use this
# file to give DocIndexer additional spelling corrections that should be
# performed on the token stream, but that are not contained in the default list
```

```
# of spelling corrections (found in the config folder).  This file can be used
# to remove or 'normalize' domain-specific slang.  The file name does not have
# to be 'user_spelling.txt'.  To NOT use a user spelling file,
# comment the next line.

#user_spelling_file: /path/to/user_spelling.txt

###############################################################################
#                                                                             #
#                            Boolean flags (Yes/No)                           #
#                                                                             #
###############################################################################

# [OPTIONAL - CUSTOM ANALYZERS ONLY] Whether to discard all tokens tagged with
# the <NUM> tag by the tokenizer.

IGNORE_NUMERALS: No

# [OPTIONAL - CUSTOM_FORMAL ANALYZER ONLY] Whether to discard all tokens except
# those tagged with the <ALPHANUM> tag.  Overrides the IGNORE_NUMERALS option.

IGNORE_ALL_BUT_ALPHANUM: No

# [OPTIONAL - CUSTOM ANALYZERS ONLY] Whether to disable stemming.  Stemming
# removes characters from selected tokens, which may not be desirable in all
# scenarios.

DISABLE_STEMMING: No

# [OPTIONAL] Whether to disable all filters in the CUSTOM analyzers.  If
# this option is selected, tokenization is the only operation performed on the
# data.  Use this option if you want to see the full set of tokens that emerge
# from the tokenizer.  This option does not apply to the Lucene-provided
# analyzers.

DISABLE_CUSTOM_FILTERS: No

<end of file>
```

The ‘config_path’, ‘outdir’, and ‘indir’ entries must all be directories, not files.

The ‘user’ files exist to augment the default sets of stop words and spelling corrections provided by DocIndexer. The default sets are small rather than large, to keep the impact on the text minimal. Some users may prefer this; other users may want large stop word or spelling correction lists. To disable the use of a user file, simply comment the appropriate line in the config file. The default configuration is to not use user files.

Users can also alter the default files located in the config folder. Stop words and spell correction mappings can be added or removed. If the default files are deleted or have no content the DocIndexer will simply ignore them.

If you find the default list of stop words and spell correction mappings to be inadequate, you can augment these default files with additional entries that should be applied to all data sets. The ‘user’ stopword file could then be used for removal of specific tokens or words relevant to the particular data set being examined. For instance, an analysis of documents about Justin Bieber could benefit from removing the token ‘bieber’ from the analysis, since it would likely appear as a top term in many (if not most) of the clusters. This can be achieved by making ‘bieber’ an entry in the user stop word file.

This file also contains several Boolean flags that can be used to selectively remove tokens from the stream. These should be self-explanatory. The DISABLE_CUSTOM_FILTERS tag will allow tokenization of the input text but no filtering. Set this flag to ‘Yes’ if you only want to see the raw token stream at the output of the tokenizer. This option is useful for investigating the effects of the analyzer on the various tokens via before-and-after comparisons in analysis mode.

IV.3.4 The Default Config File for Twitter

To process Twitter JSON data stored in a filesystem hierarchy (the files must have a `.json` extension), start with the config file called '`default_twitter_config.yml`'. This file is stored in the top-level LEAN folder. Make a copy of this file, rename it, and then set the paths appropriately for your system. Here is a listing of the contents of this file, followed by a discussion of the various items:

File 'default_twitter_config.yml':

```
################################################################################
#                                                                              #
#                       GENERAL CONFIGURATION DETAILS                          #
#                                                                              #
#     Configure DocIndexer to processes Twitter JSON data contained in         #
#     flat files.                                                              #
#                                                                              #
################################################################################

# [REQUIRED] Specify the full path to the LEAN repository's 'config' folder.
# This folder contains default stopword and spelling files, among other items.
# Items in this folder are loaded automatically at startup.

config_path: /path/to/lean/config

# [REQUIRED] Write the Lucene index to this folder.  If this folder does not
# exist it will be created.

outdir: /path/to/index

# [REQUIRED] Specify the analyzer to use.  An analyzer consists of a tokenizer
# and a chain of zero or more token filters.  The filters perform various
# transformations on the tokens as they pass down the chain.  The first four
# analyzers are provided by Lucene and are not customizable; the 'custom'
# analyzers can be easily altered and recompiled.


# org.apache.lucene.analysis.core.WhitespaceAnalyzer      Split text on whitespace only
# org.apache.lucene.analysis.standard.StandardAnalyzer    Lucene's default text analyzer
# org.apache.lucene.analysis.standard.ClassicAnalyzer     Lucene's StandardAnalyzer pre v3.1
# org.apache.lucene.analysis.en.EnglishAnalyzer           Lucene's English-specific text analyzer
# analyzers.FormalAnalyzer                                GTRI analyzer for formal documents
# twitter.TwitterAnalyzer                                 GTRI custom analyzer for Twitter

analyzer: twitter.TwitterAnalyzer

# [REQUIRED] Specify the full path to the root folder of the file tree to be
# analyzed.  DocIndexer will begin at this root folder and recursively process
# all documents in supported formats throughout the tree.

indir: /path/to/input_folder_root

# [OPTIONAL] Specify the absolute path to a 'user' stopword file.  Use this
# file to give DocIndexer additional stopwords that should be removed from the
# token stream, but that are not contained in the default stopword list
# (found in the config folder).  This file can also be used to remove specific
# tokens that may not be of interest for a given data set.  The file name does
# not have to be 'user_stopwords.txt'.  To NOT use a user stopword file,
# comment the following line.

#user_stopword_file: /path/to/user_stopwords.txt

# [OPTIONAL] Specify the absolute path to a 'user' spelling file.  Use this
# file to give DocIndexer additional spelling corrections that should be
# performed on the token stream, but that are not contained in the default list
# of spelling corrections (found in the config folder).  This file can be used
# to remove or 'normalize' domain-specific slang.  The file name does not have
# to be 'user_spelling.txt'.  To NOT use a user spelling file,
# comment the next line.

#user_spelling_file: /path/to/user_spelling.txt
```

```
################################################################################
#                                                                              #
#                            Boolean flags (Yes/No)                            #
#                                                                              #
################################################################################

# [OPTIONAL] Whether to ignore retweets.  Retweets have "RT" as the first
# token in the "text" field.

IGNORE_RETWEETS: No

# [OPTIONAL] Whether to ignore hashtags, i.e. tokens such as #justinbieber.
# NOTE: this will ignore only those hashtags identified with the '#' tag by
# the part-of-speech tagger.  Hashtags used as proper nouns (^) or in other
# meaningful ways will be kept.

IGNORE_HASHTAGS: No

# [OPTIONAL] Whether to ignore URLs and e-mail addresses.  Enabling this flag
# removes all tokens having a 'U' tag.

IGNORE_URLS: No

# [OPTIONAL] Whether to ignore at-mentions, i.e. long strings of numbers that
# begin with an '@' symbol.  Enabling this flag removes all tokens having an
# '@' tag.

IGNORE_AT_MENTIONS: No

# [OPTIONAL] Whether to ignore numbers, dates, times, etc.  Enabling this flag
# removes all tokens having a '$' tag.

IGNORE_NUMERALS: No

# [OPTIONAL] Whether to disable stemming in the CUSTOM analyzers.

DISABLE_STEMMING: No

# [OPTIONAL] Whether to disable all filters in the CUSTOM analyzers.  If
# this option is selected, tokenization is the only operation performed on the
# data. Use this option if you want to see the full set of tokens that emerge
# from the tokenizer.

DISABLE_CUSTOM_FILTERS: No

<end of file>
```

The 'config_path', 'outdir', and 'indir' entries must all be directories, not files. The same comments apply here to the optional 'user' files as in the config for general documents.

IV.3.5 Example Mongo ingest configuration

To process Twitter data stored in a MongoDB database, start with the config file called '`default_mongo_config.yml`'. This file is stored in the top-level LEAN folder. Make a copy of this file, rename it, and then set the paths appropriately for your system. Here is a listing of the contents of this file, followed by a discussion of the various items:

File 'default_mongo_config.yml':

```
################################################################################
#                                                                              #
#                        GENERAL CONFIGURATION DETAILS                         #
#                                                                              #
#     Configure DocIndexer to processes Twitter JSON data contained in         #
#     a mongo database.                                                        #
#                                                                              #
```

```
################################################################################

# [REQUIRED] Specify the full path to the LEAN repository's 'config' folder.
# This folder contains default stopword and spelling files, among other items.
# Items in this folder are loaded automatically at startup.

config_path: /path/to/lean/config

# [REQUIRED] Write the Lucene index to this folder.  If this folder does not
# exist it will be created.

outdir: /path/to/index

# [REQUIRED] Specify the analyzer to use.  An analyzer consists of a tokenizer
# and a chain of zero or more token filters.  The filters perform various
# transformations on the tokens as they pass down the chain.  The first four
# analyzers are provided by Lucene and are not customizable; the 'custom'
# analyzers can be easily altered and recompiled.

# org.apache.lucene.analysis.core.WhitespaceAnalyzer      Split text on whitespace only
# org.apache.lucene.analysis.standard.StandardAnalyzer    Lucene's default text analyzer
# org.apache.lucene.analysis.standard.ClassicAnalyzer     Lucene's StandardAnalyzer pre v3.1
# org.apache.lucene.analysis.en.EnglishAnalyzer           Lucene's English-specific text analyzer
# analyzers.FormalAnalyzer                                GTRI analyzer for formal documents
# twitter.TwitterAnalyzer                                 GTRI custom analyzer for Twitter

analyzer: twitter.TwitterAnalyzer

# [REQUIRED FOR MONGO INPUT] access the mongo instance available at this host address
host: localhost

# [REQUIRED FOR MONGO INPUT] access the mongo instance served at this port
port: 27019

# [REQUIRED FOR MONGO INPUT] connect to this database
database: twitter

# [REQUIRED FOR MONGO INPUT] ingest tweets from this collection
collection: ows

# [REQUIRED FOR MONGO INPUT] collect up to this many tweets;
# using a value of -1 will result in no limit applied to the tweets collected
limit: -1

# [OPTIONAL] Specify the absolute path to a 'user' stopword file.  Use this
# file to give DocIndexer additional stopwords that should be removed from the
# token stream, but that are not contained in the default stopword list
# (found in the config folder).  This file can also be used to remove specific
# tokens that may not be of interest for a given data set.  The file name does
# not have to be 'user_stopwords.txt'.  To NOT use a user stopword file,
# comment the following line.

#user_stopword_file: /path/to/user_stopwords.txt

# [OPTIONAL] Specify the absolute path to a 'user' spelling file.  Use this
# file to give DocIndexer additional spelling corrections that should be
# performed on the token stream, but that are not contained in the default list
# of spelling corrections (found in the config folder).  This file can be used
# to remove or 'normalize' domain-specific slang.  The file name does not have
# to be 'user_spelling.txt'.  To NOT use a user spelling file,
# comment the next line.

#user_spelling_file: /path/to/user_spelling.txt

################################################################################
#                                                                              #
#                            Boolean flags (Yes/No)                            #
#                                                                              #
################################################################################

# [OPTIONAL] Whether to ignore retweets.  Retweets have "RT" as the first
# token in the "text" field.

IGNORE_RETWEETS: No
```

```
# [OPTIONAL] Whether to ignore hashtags, i.e. tokens such as #justinbieber.
# NOTE: this will ignore only those hashtags identified with the '#' tag by
# the part-of-speech tagger.  Hashtags used as proper nouns (^) or in other
# meaningful ways will be kept.

IGNORE_HASHTAGS: No

# [OPTIONAL] Whether to ignore URLs and e-mail addresses.  Enabling this flag
# removes all tokens having a 'U' tag.

IGNORE_URLS: No

# [OPTIONAL] Whether to ignore at-mentions, i.e. long strings of numbers that
# begin with an '@' symbol.  Enabling this flag removes all tokens having an
# '@' tag.

IGNORE_AT_MENTIONS: No

# [OPTIONAL] Whether to ignore numbers, dates, times, etc.  Enabling this flag
# removes all tokens having a '$' tag.

IGNORE_NUMERALS: No

# [OPTIONAL] Whether to disable stemming in the CUSTOM_TWITTER analyzer.

DISABLE_STEMMING: No

# [OPTIONAL] Whether to disable all filters in the CUSTOM_TWITTER analyzer; if
# this option is selected, tokenization is the only operation performed on the
# data. Use this option if you want to see the full set of tokens that emerge
# from the tokenizer.

DISABLE_CUSTOM_FILTERS: No

<end of file>
```

IV.4 Analyzers

Lucene "analyzers" consist of a tokenizer followed by zero or more token filters. The filters perform specific transformations on the tokens, such as conversion to lowercase. Lucene provides a large number of analyzers, both for English and other languages. It also allows the creation of new analyzers with custom filter chains.

IV.4.1 Default Lucene Analyzers

The following four Lucene analyzers are available to DocIndexer and provide helpful baselines for evaluating custom analyzer performance. Select the desired analyzer by specifying it within the configuration file. More information on each of these analyzers can be obtained from the official Lucene[13] documentation.

org.apache.lucene.analysis.core.WhitespaceAnalyzer
The WhitespaceAnalyzer is the most basic analyzer, simply separating tokens based on whitespace.

org.apache.lucene.analysis.standard.StandardAnalyzer
The StandardAnalyzer uses a JFlex-based tokenizer and applies standard, lowercase, and stopword filters. The StandardAnalyzer implements the word break rules from the Unicode text segmentation algorithm of Unicode Standard Annex 29.

org.apache.lucene.analysis.standard.ClassicAnalyzer
The ClassicAnalyzer is an earlier version (Lucene version 3.1) of the StandardAnalyzer. The behavior of this analyzer conforms to that of the StandardAnalyzer prior to the implementation of the Unicode Standard Annex 29 word break rules.

org.apache.lucene.analysis.en.EnglishAnalyzer
The EnglishAnalyzer is Lucene's English-specific analyzer and, in addition to the filters used in the StandardAnalyzer, applies possessive, keyword, and stemming (Porter stemmer) filters. It can also change word endings of various words, which may not be desirable for some applications.

IV.4.2 FormalAnalyzer

The LEAN FormalAnalyzer (located in the file `DocIndexer/src/analyzers/FormalAnalyzer.java`) is a customized chain of token filters specific for formal English documents, such as news articles or reports. The filters below are included in the chain of filters and can either be rearranged or selectively excluded (simply comment out the appropriate '`chain =`' line in the source code) as desired. Additional filters from Lucene can also be inserted into the chain. We suggest exploration of which filters and which order best extracts meaningful tokens from your data. Use of the `--analyze` option with the DocIndexer application can let you see the effect that these changes have on the extracted tokens.

StandardFilter
This filter does nothing (at present); it is traditionally used as a first filter before the additional chain of more specific filters.

StandardTagFilter (`DocIndexer/src/filters/StandardTagFilter.java`)
This filter removes tokens having specific tags (such as <NUM>) from the token stream. The Boolean options in the config file control the operation of this filter.

[13] lucene.apache.org

LowerCaseFilter
This filter converts text to its lowercase equivalent.

ASCIIFoldingFilter
This filter converts alphabetic, numeric, and symbolic Unicode characters which are not in the first 127 ASCII characters (the "Basic Latin" Unicode block) into their ASCII equivalents, if such exists.

StopFilter
This filter removes common words from the token stream.

EnglishPossessiveFilter
This filter removes the possessive ‘s from tokens.

SpellingCorrectionFilter (`DocIndexer/src/filters/SpellingCorrectionFilter.java`)
This filter performs a hashtable lookup on tokens to identify and correct commonly misspelled words. The hashtable is composed of all mappings in the default spelling file and the optional user-provided spelling file.

KStemFilter
This filter is an implementation of the Krovetz stemmer, which was developed by Bob Krovetz at the University of Massachusetts in 1993. It is a 'light' stemmer, as it makes use of inflectional linguistic morphology. An alternative, and more aggressive, stemmer that could be used is the PorterStemFilter.

IV.4.3 TwitterAnalyzer

The analysis of informal text (such as Twitter tweets, web chat, and SMS messages) presents some different challenges from the analysis of formal text. We have selected a series of Lucene-provided and custom token filters to create a TwitterAnalyzer (located at `DocIndexer/src/twitter/TwitterAnalyzer.java`) that handles these challenges more effectively than the standard Lucene analyzers. Some of the filters from the FormalAnalyzer were used, in addition to the custom filters identified below.

We have leveraged the work of the NLP research group at Carnegie Mellon University by incorporating their Twitter-aware tokenizer[14] into our custom analyzer. This tokenizer also includes a part-of-speech tagger that performs well on Twitter text; the model for the tagger was trained using statistical methods on a large corpus of tweets.

NOTE: the Twitter analyzer will only process files having a .json extension.

As with the FormalAnalyzer, we suggest exploration of which filters and which order best extracts meaningful tokens from your specific corpus of informal text. Use of the `--analyze` option with the DocIndexer application can let you see the effect that these changes have on the extracted tokens. The Twitter analyzer includes one or more instances of the following filters:

TweetNLPTagFilter (`DocIndexer/src/filters/TweetNLPTagFilter.java`)
This filter removes tokens identified as meaningless by the CMU TweetNLP part-of-speech tagger. Any token tagged with the '&', '~', 'E', '!', ',', or 'G' tags is discarded from the token stream. The boolean flags in the config files allow for selective removal of tokens tagged with the '#', 'U', '$', and '@' tags. See the '`TweetNLP.pdf`' document in the lean/doc folder for a description of what these tags mean. In the source code, the auto-removal tags appear in the `DEFAULT_TAGSET` array at the top of the filter file.

LowercaseFilter
This filter is provided by Lucene and simply converts all text to lowercase.

AsciiFoldingFilter
This Lucene filter converts alphabetic, numeric, and symbolic Unicode characters which are not in the first 127 ASCII characters (the "Basic Latin" Unicode block) into their ASCII equivalents, if such exists.

StopFilter
This filter is provided by Lucene; it removes stop words from both the default stopword list and the optional user stop word list.

RepeatedCharFilter (`DocIndexer/src/filters/RepeatedCharFilter.java`)
This filter replaces all occurrences of three or more characters by a two-character sequence. For example, "hellllllllllo" would become "hello" after application of this filter. Character repetitions such as this are common in informal text. Without this filter the tokens 'hello', 'helllo', and 'hellllo' would be entered into the dictionary as separate terms, even though they are just variants on the basic 'hello'.

[14] http://www.ark.cs.cmu.edu/TweetNLP/

UnicodeCleanupFilter (`DocIndexer/src/filters/UnicodeCleanupFilter.java`)
This filter removes a variety of potentially useless tokens and corrupted text that is sometimes attached to tokens. These include:

- Replacing all Unicode dash codepoints with the ASCII hyphen/minus;
- Removing all explicit Unicode punctuation codepoints;
- Replacing any backquote characters with an ASCII apostrophe
- Replacing currency expressions with the currency amount, i.e. US$500 -> $500
- Removing all emoji codepoints;
- Removing all 'private use' codepoints;

TokenSplitterFilter (`DocIndexer/src/filters/TokenSplitterFilter.java`)
This filter splits compound tokens containing separating punctuation into individual tokens and adds them as individual tokens in the token stream. For instance, this filter will expand the token 'food/materials' into the separate tokens 'food' and 'materials'. The original compound token is marked for discard later in the CompundRemovalFilter to ensure no unnecessary duplication. Such compound tokens occur frequently in twitter data.

CompoundRemovalFilter (`DocIndexer/src/filters/CompoundRemovalFilter.java`)
This filter removes compound tokens marked for removal by the TokenSplitterFilter.

TrimFilter (`DocIndexer/src/lucenetools/TrimFilter.java`)
This filter strips leading and trailing punctuation and other characters from the tokens.

SlangNormalizationFilter (`DocIndexer/src/filters/SlangNormalizationFilter.java`)
This filter performs a hashtable lookup on tokens to identify slang, such as 'b4' used to represent 'before'; 'prez' to represent 'president', etc. Any slang thus identified is replaced by its standard English equivalent (this process is called *normalization*). The mapping of standard English to slang is contained in the 'default_slang.txt' file in the config directory.

EnglishPossessiveFilter
This is a Lucene-provided filter that removes possessive endings from tokens.

SpellingCorrectionFilter (`DocIndexer/src/filters/SpellingCorrectionFilter.java`)
This filter performs a hashtable lookup on tokens to identify and correct commonly misspelled words. The hashtable is composed of all mappings in the default spelling file and the optional user-provided spelling file.

KStemFilter
This Lucene-provided filter performs Krovetz stemming on the tokens.

V. Usage of LuceneToMtx

V.1 Overview

The LuceneToMtx application reads the Lucene index generated from the DocIndexer application and converts it into a term-frequency `matrix.mtx` file, along with a `documents.txt` file and a `dictionary.txt` file.

Note: The LuceneToMtx application requires a Lucene index that has 'term vectors' enabled. An arbitrary Lucene index will not necessarily be compatible.

V.2 Command Line Application

LuceneToMtx is an executable jar file. If you run the jar file without any command-line arguments, or with the `-h` or `--help` arguments, the application will output the command-line options supported. For instance, from the top-level LEAN folder, display the supported command-line options by running this command:

```
java -jar LuceneToMtx/dist/LuceneToMtx.jar
```

You should see the following output:

```
Usage: java -jar /path/to/LuceneToMtx.jar
                --indir      <path to Lucene index dir>
               [--outdir]    result files written here
               [--minfreq 1] minimum term frequency cutoff value
               [--maxprct 100] maximum term occurrence
               [--fields 0] write out all fields in the index
```

Only the first parameter, `--indir`, is required. The meanings of the parameters are as follows:

1. `--indir`: File path to the directory containing the Lucene index generated by the DocIndexer application.

2. `--outdir`: File path to desired output directory. If unspecified, the output files will be written to the current directory. This directory will be created if it does not exist.

3. `--minfreq`: Minimum value for term frequency in the generated term frequency matrix. Terms will only be included in the output matrix if their frequency count is greater than or equal to this cutoff value. By setting this parameter to a value greater than 1, infrequently-occurring terms and noise can be pruned from the term frequency matrix, thus making the *smallk* preprocessor's work easier, and reducing the required I/O time. For Twitter data we see significant cleanup of noisy tokens by using a value of 15 or greater.

4. `--maxprct`: Maximum term occurrence compared to the entire corpus. Terms will only be included in the generated term frequency matrix if their overall percentage is less than this value. This can be used to ensure that common and very frequent terms that contribute little semantic value are excluded. The flip side of this, of course, is that terms that you really are interested in can inadvertantely be removed.

5. `--fields`: Write out all non-required fields found in the index. All non-required fields that have been added to the Lucene index via the DocIndexer application will be written to files as well as the default set of output documents. This is helpful for when there is value in tracking additional features of the text, such as latitude and longitude points for tweets.

The LuceneToMtx application outputs a `matrix.mtx` file, a `dictionary.txt` file, and a `documents.txt` file (as well as whatever other fields may be written out by setting the fields parameter to 1). These files can be used as input to the SmallK[15] suite of document clustering tools.

Note: Depending on the size of the index being processed, you may find it necessary to increase the amount of memory allocated to the JVM. You can do so by specifying the following two parameters on the command line when running the application.

```
-Xms<size>     specifies the initial Java heap size (default is 2Mb)
-Xmx<size>     specifies the maximum Java heap size (default is 64Mb)
```

For example, to set the minimum at 64Mb and the maximum at 256Mb:

```
java -Xms64m -Xmx256m -jar dist/LuceneToMtx.jar
```

LuceneToMtx prints out the maximum available memory when it launches. You should see this value update with a new value close to that specified in the script.

However, since LuceneToMtx is built and bundled into an executable jar file, this method may not work if you run this command directly in your terminal. If it does not work on your system, copy this command line into a shell script, then execute the shell script.

For instance, to use the shell script method, create a new file called run.sh, copy the command line above into it, and then execute the script with this command:

```
sh run.sh
```

You should then see LuceneToMtx report a max memory value similar to what you specify.

As a usage example, suppose you run the DocIndexer application and generate a Lucene index in the folder `lean/DocIndexer/index`. To process this index with LuceneToMtx and write the result files to the folder `lean/LuceneToMtx/results`, run the following command from the top-level LEAN folder:

```
java -jar LuceneToMtx/dist/LuceneToMtx.jar --indir DocIndexer/index --outdir LuceneToMtx/results
--minfreq 10
```

This command will generate a term-frequency matrix that includes only those terms occurring 10 or more times in the input corpus.

V.3 Output Files

LuceneToMtx generates at least three output files: `dictionary.txt`, `documents.txt`, and `matrix.mtx`.

[15] http://smallk.github.io/

The dictionary file is a simple list of strings, one per line. These are the tokens that survived the filtering and pruning operations of DocIndexer and LuceneToMtx. Each string in this file is a 'term' in a 'dictionary', which this file represents.

The documents file is also a simple list of strings, one per line. The entries in this file are the documents that survived the filtering and pruning operations of DocIndexer and LuceneToMtx. For twitter data, the strings have the following format: `<filename>_<tweet index>`, where 'tweet_index' is the 0-based numeric index of the tweet in the file.

The matrix file is a sparse matrix in MatrixMarket format (the official format document is stored in the LEAN doc folder as `MMformat.pdf`). The number of rows of the matrix equals the number of terms in the dictionary file. The number of columns in the matrix equals the number of document strings in the document file.

Any other files that are written out contain fields that were indexed as part of the DocIndexer application. For example, if geo information is extracted from Tweets, a `geo.txt` file could contain one line per document that contains the latitude and longitude for that Tweet.

VI. Acquiring Test Data

VI.1 Included and Recommended Datasets

Five datasets are included in the LEAN repository for demonstration and testing purposes.

- `path/to/LEAN/data/news_articles`: 200 news articles from the Australian Broadcasting Commission.
- `path/to/LEAN/data/twitter_test`: 55 problematic tweets from the Occupy Wall Street dataset.
- `path/to/LEAN/data/html_files`: HTML Wikipedia pages on various philosophers organized in hierarchical folders.
- `path/to/LEAN/data/pdf_files`: Amazon Web Services White Papers in pdf form.
- `path/to/LEAN/data/mongo.zip`: A compressed MongoDB collection containing 500 Occupy Wall Street tweets.

We recommend downloading the full corpus of Occupy Wall Street tweets[16] (~50k) to use for testing purposes. Click on the 'Download Code Examples (ZIP)' link and download the file `TDA.zip`. Unzip it somewhere on your system. You should see a `Data/` folder, among other things. The file `ows.json` holds 28 MB of Twitter data, containing 49,147 tweets about the Occupy Wall Street protests.

Any folder (or hierarchy of folders) can be used as the input to DocIndexer using the file system ingest configuration. DocIndexer will recurse through the file tree and process all files in recognized formats.

[16] http://tweettracker.fulton.asu.edu/tda

VI.2 Using MongoDB Databases

The Twitter API exports data in JSON format, making MongoDB a natural choice for receiving and storing this data. The LEAN repository includes a Mongo collection populated with 500 tweets. To access this collection, you must first install MongoDB.

For MacOSX, you can install MongoDB via Homebrew:

```
brew install mongo
```

For Linux, you can use a package manger:

```
apt-get install mongo
```

Uncompress the Mongo database files, which are in `data/mongo.zip`. Change the name of the '`mongo_files`' directory if you wish:

```
cd path/to/LEAN/data
unzip mongo.zip -d mongo_files
```

For Linux, you may need to install `unzip` to run the above command:

```
apt-get install unzip
```

In one terminal, start the Mongo server using the database files provided. We specify a non-standard port to ensure no port conflicts with other Mongo servers.

```
mongod --dbpath path/to/LEAN/data/mongo_files/mongo --port 27019
```

Copy the `default_mongo_config.yml` configuration file, rename it, and edit it so that the 'config_path' and 'outdir' variables are appropriate for your system. The remaining fields are set up to use the Mongo server we started on port 27019. In a different terminal, you can now run DocIndexer using your new configuration file.

VIII. Clustering Example using Smallk

In this section we provide a step-by-step description of how to use the LEAN tools together with SmallK to cluster a set of documents. The documents to be clustered are the set of tweets related to the Occupy Wall Street protests, contained in the `ows.json` file. Download instructions for this dataset can be found in section six of this document.

We will refer to the top-level LEAN folder as LEAN_DIR, and the top-level SmallK folder (smallk project) as SMALLK_DIR. The user's home directory will be denoted by HOME. We assume that both projects have been cloned and built as described in the documentation for each project.

0: Setup and configuration

Create a temporary folder into which output files will be written.

```
mkdir -p ~/tmp/lean_demo
```

Setup a LEAN configuration file.

```
cd LEAN_DIR
cp default_twitter_config.yml my_config.yml
```

Edit the file `my_config.yml` so that the following entries are uncommented and have the values specified. Substitute the appropriate absolute paths on your system for HOME and LEAN_DIR:

```
config_path: /path/to/LEAN_DIR/config
outdir: HOME/tmp/lean_demo/index
analyzer: CUSTOM_TWITTER
indir: /path/to/ows/directory
IGNORE_HASHTAGS: No
IGNORE_URLs: No
IGNORE_AT_MENTIONS: No
IGNORE_NUMERALS: No
DISABLE_STEMMING: No
DISABLE_CUSTOM_FILTERS: No
```

This will configure LEAN to use the file 'ows.json' on your system as input, and to write the Lucene index to `~/tmp/lean_demo/index`.

1. Use the LEAN DocIndexer application in analyze mode.

Test different analyzers and configurations.

Let's first see what the tokens look like with only tokenization. Edit the configuration file to show:

```
DISABLE_CUSTOM_FILTERS: Yes
```

Run DocIndexer and look at the output of the first few tweets:

```
cd LEAN_DIR
java -jar DocIndexer/dist/DocIndexer.jar --config my_config.yml --analyze 5
```

You will see output similar to the following:

```
DocIndexer version 0.1.
...
Analysis results for file ../samples.json, 5 tweets max:

[1]     RT @2141912560879618632: Impeach Scalia and Thomas.  Clarence Thomas also a tax
fraud.  #p2 http://t.co/SuzLE7kZ
start:    0    end:    2      length:    2   tag: ~  token: RT
start:    3    end:   23      length:   20   tag: @  token: @2141912560879618632
start:   23    end:   24      length:    1   tag: ~  token: :
start:   25    end:   32      length:    7   tag: V  token: Impeach
start:   33    end:   39      length:    6   tag: ^  token: Scalia
start:   40    end:   43      length:    3   tag: &  token: and
start:   44    end:   50      length:    6   tag: ^  token: Thomas
start:   50    end:   51      length:    1   tag: ,  token: .
start:   53    end:   61      length:    8   tag: ^  token: Clarence
start:   62    end:   68      length:    6   tag: ^  token: Thomas
start:   69    end:   73      length:    4   tag: R  token: also
start:   74    end:   75      length:    1   tag: D  token: a
start:   76    end:   79      length:    3   tag: N  token: tax
start:   80    end:   85      length:    5   tag: N  token: fraud
start:   85    end:   86      length:    1   tag: ,  token: .
start:   88    end:   91      length:    3   tag: #  token: #p2
start:   92    end:  112      length:   20   tag: U  token: http://t.co/SuzLE7kZ

...
```

You can see that there are many useless tokens included. Edit the configuration file to enable filters and rerun the above command. You will see the following output:

```
[1]     RT @2141912560879618632: Impeach Scalia and Thomas.  Clarence Thomas also a tax
fraud.  #p2 http://t.co/SuzLE7kZ
start:    3    end:   23      length:   20   tag: @  token: @2141912560879618632
start:   25    end:   32      length:    7   tag: V  token: impeach
start:   33    end:   39      length:    6   tag: ^  token: scalia
start:   44    end:   50      length:    6   tag: ^  token: thomas
start:   53    end:   61      length:    8   tag: ^  token: clarence
start:   62    end:   68      length:    6   tag: ^  token: thomas
start:   76    end:   79      length:    3   tag: N  token: tax
start:   80    end:   85      length:    5   tag: N  token: fraud
start:   88    end:   91      length:    3   tag: #  token: #p2
start:   92    end:  112      length:   20   tag: U  token: http://t.co/suzle7kz
...
```

The filters have improved the results by discarding retweet indicators and punctuation. You can modify other fields in the configuration file or even the code itself and experiment with the tokens produced.

Iterate this step until you are satisfied that the tokens generated look clean and reasonable.

2. Generate a term-frequency matrix and accompanying files.

Use DocIndexer in normal mode.

Once you are satisfied with your filter configuration, use DocIndexer in normal mode to generate a Lucene index.

```
java –jar DocIndexer/dist/DocIndexer.jar --config my_config.yml
```

The `ows.json` data file contains 49,147 tweets. With all filters enabled, a mid-2012 MacBook Pro with a 2.6 GHz Intel Core i7 processor can process the file at ~750 tweets per second.

Use LuceneToMtx to generate a term-frequency matrix and accompanying files.

```
cd LEAN_DIR
java –jar LuceneToMtx/dist/LuceneToMtx.jar --indir ~/tmp/lean_demo/index/
      --outdir ~/tmp/lean_demo --minfreq 15
```

This generates a sparse matrix with over 450,000 nonzeros. Some of the documents (tweets) do not appear in the final matrix because of the pruning of terms with a minimum occurrence count of 15.

View the `dictionary.txt` file that was output from LuceneToMtx and review the tokens. If there are multiple spellings discovered for common tokens, words that you would like removed from the output altogether, or additional filters needed, iterate this step and possibly step 1.

For example, we observe that our dictionary includes misspellings of zuccotti:

```
...
zucatti
zuccoti
zucotti
```

We can modify our `user_spelling_file` configuration to include a file containing the correct spelling on the left and the misspellings in a CSV list on the right:

```
zuccotti    zucatti, zuccoti, zucotti
```

If we modify our configuration file to point the `user_spelling_file` variable to our newly created file and repeat the indexing process, we see that our dictionary now contains only the correctly spelled token for zuccotti:

```
...
zuc
zuccotti
```

Iterate this step until you are satisfied with the results of the `dictionary.txt` file, then move on to step 3.

3. Use SmallK to cluster the documents.

Preprocess the matrix to remove any duplicate columns.

```
mkdir ~/tmp/lean_demo/reduced
```

```
cd SMALLK_DIR
./preprocessor/bin/preprocess_tf --indir ~/tmp/lean_demo --outdir ~/tmp/lean_demo/reduced
```

This generates a new matrix in `~/tmp/lean_demo/reduced`, with a size of approximately 3,450 by 28,000 with approximately 260,000 nonzeros. Matching dictionary and document files are also written to this folder.

Perform rank2 hierarchical clustering on the document set represented by the reduced matrix.

```
mkdir ~/tmp/lean_demo/clusters
cd SMALLK_DIR
./hierclust/bin/hierclust --matrixfile ~/tmp/lean_demo/reduced/reduced_matrix.mtx
      --dictfile ~/tmp/lean_demo/reduced/reduced_dictionary.txt
      --clusters 32
      --maxterms 10
      --outdir ~/tmp/lean_demo/clusters
```

This command generates 32 clusters with 10 terms each and writes the clustering results to the `~/tmp/lean_demo/clusters` folder.

Two output files are generated by the hierclust application: assignments_32.csv and tree_32.xml. If you open the xml file in a text editor you can see the nodes of the factorization tree. The leaf nodes of the tree have the `left_child_id` and `right_child_id` both equal to -1.

You may observe that some potentially uninteresting tokens appear in the top terms for the leaf nodes. For example, some hashtags may appear in the top terms for each node, as well as 'at-mentions' (a '@' symbol followed by a string of digits).

Hashtags and at-mentions can be suppressed by setting the boolean flags in the configuration file. Try setting the IGNORE_HASHTAGS and IGNORE_AT_MENTIONS to 'Yes' and repeating steps 2 and 3 to see the effect on the resulting leaf nodes.

Individual tokens, hashtags, URLs, etc. can be suppressed by entering them in a user stopword file. For instance, if you decide to keep hashtags enabled, you can suppress the common #ows hashtag by entering it in the user stopword file.

If the config file variables and Boolean flags are not sufficient to achieve the results you want, you can always edit the code for the TwitterAnalyzer. The Krovets stemmer could be replaced by a Porter stemmer; additional filters could be applied; or the order in which the filters appear could be changed. There is no *apriori* method to know which processing works best on which data sets. We have provided a customizable and extensible framework that can hopefully be used to achieve the results you desire.

IX. Appendix

IX.1 Sample Datasets

Many sample datasets are available that can be used for testing and customizing LEAN.

Python's Natural Language Processing Toolkit (NLTK) comes with several different text corpora, which can be useful for text analytics. Download NLTK here[17]. Once you have downloaded and installed NLTK, you need to download and install the data. To do this, start a Python interpreter and enter the following commands at the prompt:

```
import nltk
nltk.download()
```

A GUI will launch on your system. Click on the 'Collections' tab, select 'all,' and download everything. After the download completes, you should see the 'nltk_data' folder in your home directory. The corpora are stored in the `corpora/` subdirectory.

IX.2 Test Scripts

We encourage use of the included tests to confirm successful installation and building of the LEAN tools. To run the full set of tests, first start a Mongo server in a terminal (setting the `dbpath` as described in the previous section):

```
mongod --dbpath path/to/LEAN/data/mongo_files/mongo --port 27019
```

In a new terminal window, launch the test script:

```
path/to/LEAN/run_tests.sh
```

If you do not wish to use MongoDB, edit the '`run_tests.sh`' script in the top-level LEAN directory and comment out the three lines that contain 'mongo'.

The script runs a series of tests, all of which are available within the `tests/` subdirectory. You should see all tests execute successfully:

```
**************************************
       SUCCESS: all tests passed
**************************************
```


[17] http://www.nltk.org/