JATE 2.0: Java Automatic Term Extraction with Apache Solr

Ziqi Zhang, Jie Gao, Fabio Ciravegna
Regent Court, 211 Portobello, Sheffield, UK, S1 4DP
ziqi.zhang@sheffield.ac.uk, j.gao@sheffield.ac.uk, f.ciravegna@sheffield.ac.uk

Abstract

Automatic Term Extraction (ATE) or Recognition (ATR) is a fundamental processing step preceding many complex knowledge engineering tasks. However, few methods have been implemented as public tools and in particular, available as open-source freeware. Further, little effort is made to develop an adaptable and scalable framework that enables customization, development, and comparison of algorithms under a uniform environment. This paper introduces JATE 2.0, a complete remake of the free Java Automatic Term Extraction Toolkit (Zhang et al., 2008) delivering new features including: (1) highly modular, adaptable and scalable ATE thanks to integration with Apache Solr, the open source free-text indexing and search platform; (2) an extended collection of state-of-the-art algorithms. We carry out experiments on two well-known benchmarking datasets and compare the algorithms along the dimensions of effectiveness (precision) and efficiency (speed and memory consumption). To the best of our knowledge, this is by far the only free ATE library offering a flexible architecture and the most comprehensive collection of algorithms.

Keywords: term extraction, term recognition, NLP, text mining, Solr, search, indexing

1. Introduction

Automatic Term Extraction (or Recognition) is an important Natural Language Processing (NLP) task that deals with the extraction of terminologies from domain-specific textual corpora. ATE is widely used by both industries and researchers in many complex tasks, such as Information Retrieval (IR), machine translation, ontology engineering and text summarization (Bowker, 2003; Brewster et al., 2007; Maynard et al., 2007). Over the years, there has been constant effort on researching new algorithms, adapting to different domains and/or languages, and creating benchmarking datasets (Ananiadou, 1994; Church and Gale, 1995; Ahmad et al., 1999; Frantzi et al., 2000; Park et al., 2002; Matsuo and Ishizuka, 2003; Sclano and Velardi, 2007; Rose et al., 2010; Spasić et al., 2013; Zadeh and Handschuh, 2014).

Given the fundamental role of ATE in many tasks, a real benefit to the community would be releasing the tools to facilitate the development of downstream applications, also encouraging code-reuse, comparative studies, and fostering further research. Unfortunately, effort in this direction has been extremely limited. First, only very few tools have been published with readily available source code (Spasić et al., 2013) and some are proprietary (labs.translated.net, 2015; Yahoo!, 2015), or no longer maintained (Sclano and Velardi, 2007)[1]. Second, different tools have been developed under different scenarios and evaluated in different domains using proprietary language resources, making it difficult for comparison. Third, it is unclear whether and how well these tools can adapt to different domain tasks and scale up to large data.

To the best of our knowledge, the only effort towards addressing these issues is the Java Automatic Term Extraction Toolkit (JATE)[2], where we originally implemented five state-of-the-art algorithms under a uniform framework (Zhang et al., 2008). This work makes one step further by completely re-designing and re-implementing JATE to fulfill three goals: **adaptability**, **scalability**, and **extended collections of algorithms**. The new library, named **JATE 2.0**[3], is built on the Apache Solr[4] free-text indexing and search platform and can be either used as a separate module, or a Solr plugin during document processing to enrich the indexed documents with terms. JATE 2.0 delivers **domain-specific adaptability** and **scalability** by seamlessly integrating with the Solr text analysis capability in the form of 'analyzers'[5], allowing users to directly exploit the highly modular and scalable Solr text analysis to facilitate term extraction for large data from various content formats. Last but not least, JATE 2.0 **expands its collection** of state-of-the-art algorithms[6] to double that in the original JATE release. We also evaluate these algorithms under uniform settings on two publicly available benchmarking datasets. Released as open source software, we believe that JATE 2.0 offers significant value to both researchers and practitioners.

The remainder of this paper is structured as follows. Section 2. discusses related work. Section 3. describes JATE 2.0 in details. Section 5. describes experiments and discusses results. Section 6. concludes this paper.

2. Related Work

We describe related work from three persectives: ATE methods in general, ATE tools and software, and text mining plugins based on Solr.

ATE methods typically start with linguistic processors as the first stage to *extract candidate terms*; followed by

[1] Original link: `http://lcl2.di.uniroma1.it/`

[2] `https://code.google.com/p/jatetoolkit`. Google Code has been shut down since Jan 2016.

[3] `https://github.com/ziqizhang/jate`

[4] `http://lucene.apache.org/solr/`

[5] `https://cwiki.apache.org/confluence/display/solr/Analyzers`

[6] By this we mean the mathematical formulation of term candidate scores. We do not claim identical replication of state-of-the-art methods, as they often differ in pre- and post-processing, such as term candidate generation, term variant handling and the use of dictionary.

statistics-based term ranking algorithms to perform *candidate filtering*. Linguistic processors are domain specific, and designed to capture term formation and collocation patterns. They often take two forms: 'closed filters' (Arora et al., 2014) focus on precision and are usually restricted to nouns or noun sequences. 'Open filters' (Frantzi et al., 2000) are more permissive and often allow adjectives, adverbs, etc. For both, widely used techniques include Part-of-Speech (PoS) tag sequence matching, N-gram extraction, Noun Phrase (NP) Chunking, and dictionary lookup. The selection of linguistic processors is often domain-specific and important for the trade-off between precision and recall. The ranking algorithms are often based on: 'unithood' indicating the collocation strength of units that comprise a single term (Matsuo and Ishizuka, 2003); and 'termhood' indicating the association strength of a term to domain concepts (Ahmad et al., 1999). Most methods combine both termhood and unithood measures (Ananiadou, 1994; Frantzi et al., 2000; Park et al., 2002; Sclano and Velardi, 2007; Spasić et al., 2013). Two unique features of JATE 2.0 are (1) customization of linguistic processors of the two forms hence making it adaptable to many domains and languages, and (2) a wide selection of implemented ranking algorithms, which are not available in any other tools.

Existing ATE tools and software are extremely limited for several reasons. First, many are proprietary software (labs.translated.net, 2015), quota-limited (e.g., OpenCalais[7]), or restricted to academic usage only (e.g., TerMine[8]). Second, all tools, including the original JATE (Zhang et al., 2008), are built in a monolithic architecture which allows very limited customization and scalability (Spasić et al., 2013). Third, most offer very limited (typically a single choice) configurability, e.g., in terms of term candidate extraction and/or ranking algorithms (e.g., TermRaider (Maynard et al., 2008), TOPIA[9], and FlexiTerm (Spasić et al., 2013). JATE 2.0 is a free, open-source library. It is highly modular, allowing customization and configuration of many components. Meanwhile, to our best knowledge, no available ATE tools support automatic indexing.

Solr plugins for text mining Solr is an open source enterprise search platform written in Java. It is highly modular, customizable, and scalable, hence widely used as an industry standard. A core component of Solr is its powerful, extensible text processing library covering a wide range of functionality and languages. Many text mining and NLP tools have integrated with Solr to benefit from these features as well as contributing additional support for document indexing (e.g., OpenNLP plugins[10], SolrTextTagger[11], and UIMA[12]). However, no plugin is available for ATE.

[7] http://new.opencalais.com/

[8] http://www.nactem.ac.uk/software/termine/

[9] https://pypi.python.org/pypi/topia.termextract

[10] https://wiki.apache.org/solr/OpenNLP

[11] https://github.com/OpenSextant/SolrTextTagger

[12] https://uima.apache.org

3. JATE 2.0 Architecture

Figure 1 shows the general architecture of JATE 2.0 and its workflow, which consists of four phases: (1) data preprocessing; (2) term candidate extraction and indexing; (3) candidate scoring and filtering and (4) final term indexing and export.

Data pre-processing (Section 3.1.) parses input documents to raw text content and performs text normalization in order to reduce 'noise' in irregular data. The preprocessed text content then passes through the **candidate extraction** (Section 3.2.) component that extracts and normalizes term candidates from each document. Candidate extraction and data pre-processing are embedded as part of the Solr document indexing process, largely realized by Solr's 'analyzers'. An analyzer is composed of a series of processors ('tokenizers' and 'filters') to form a pipeline, or an analysis 'chain' that examines the text content from each document, generates a token stream and records statistical information. Solr already implements a very large text processing library for this purpose. JATE 2.0 further extends these. Depending on individual needs, users may assemble customized analyzers for term candidate generation. Defining analyzers is achieved by configuration in the Solr schema, such as that shown in Figure 2.

Next, the candidates are processed by the subsequent **filtering** component (Section 3.3.), where different ATE algorithms can be configured to score and rank the candidates, and making the final selection. The implementation of the algorithms are largely parallelized. The resulting terms can either be exported for further validation, or written back to the index to annotate documents.

JATE 2.0 can be run in two modes: (1) as a separate module to extract terms from a corpus using an embedded Solr instance (Section 4.1.), or (2) as a Solr plugin that performs ATE as part of the document indexing process to annotate documents with domain-specific terminologies (Section 4.2.). Both modes undergo the same workflow described above.

JATE 2.0 offers adaptability and scalability thanks to its seamless integration with Solr and parallelized implementation. Access to Solr's very large collection of multilingual text processing libraries that can be configured in a plug-and-play way enables JATE 2.0 to be adapted to different languages, document formats, and domains. The rich collection of algorithms also give users options to better tailor the tool to specific use cases. Parallelization enables JATE 2.0 to scale up to large datasets. Parallelization for term candidate generation can be achieved via Solr's distributed indexing capability under SolrCloud[13]. Users simply follow Solr's guide on configuring distributed indexing. Parallelization for scoring and ranking algorithms comes as standard in JATE 2.0 and is also configurable. Detailed guidelines on defining analyzers for ATE and configurations can be found on the JATE 2.0 homepage.

[13] https://cwiki.apache.org/confluence/display/solr/SolrCloud

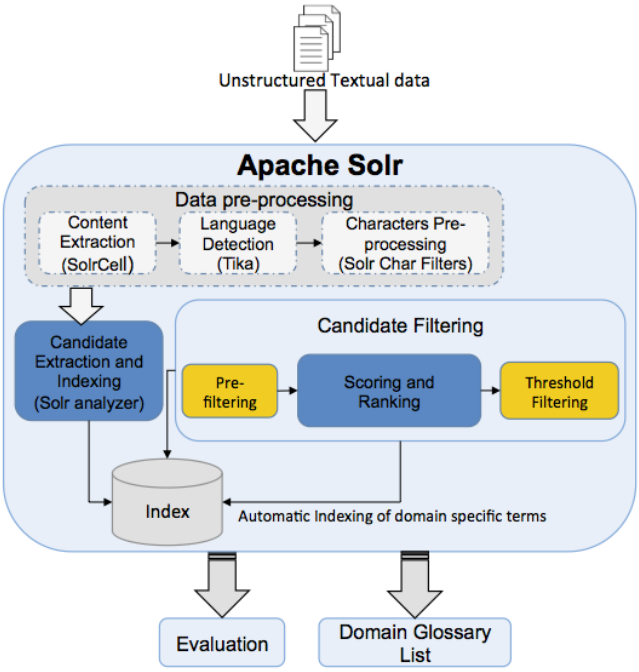

Figure 1: General Architecture of JATE 2.0

3.1. Data pre-processing

With the standard JATE 2.0 configuration, data pre-processing begins with Solr Content Extraction Library (also known as SolrCell[14]) to extract textual content from files. It integrates the popular Apache Tika framework[15] that supports detecting and extracting metadata and text from a large range of file formats such as plain text, HTML, PDF, and Microsoft Office. To use this, users simply configure an instance of `ExtractingRequestHandler`[16] for their data in Solr.

To support multi-lingual documents, users may also use Solr's capability for detecting languages of texts in documents and map text to language-specific analyzers for indexing using the `langid UpdateRequestProcessor`[17].

Next, a recommended practice for pre-processing irregular textual data (commonly found in corporate datasets) is applying the Solr char filter component[18] to perform character-level text normalization. For example, `HTMLStripCharFilterFactory` detects HTML entities (e.g., 'A', ' ') with corresponding characters. This can reduce errors in downstream processors (e.g., PoS tagger) in the analyzer. This can be configured as part of the analyzer in the Solr schema, usually placed before tokenization, such as line 3 of schema.xml shown in Figure 2.

3.2. Candidate Extraction

Term candidate extraction is realized as part of the Solr indexing process. A dedicated 'field'[19] is defined within the Solr indexing schema to hold candidates for each document (*candidate field*). The field is bound to a 'fieldType', which uses a customized analyzer to break content into term candidates and normalize them to conflate the variants that are dispersed throughout the corpus. The Solr indexing process also generates *basic statistical information* about candidates, such as a candidate's frequency within each document, and the number of documents it appears in[20].

The analyzer is highly customizable and adaptable. All existing Solr text analysis libraries can be used, such as tokenization, case folding, stop words removal, token N-gram extraction, stemming, etc. It is also easy to implement new components to replace or complement existing ones in a plug-and-play way. Currently, Solr 5.3 supports nearly 20 implementations of tokenization and over 100 text normalization and filtering methods[21], covering a wide range of languages and use cases. Thus by selecting different components and assembling them in different orders, one can create different analyzers to achieve different extraction and normalization results.

To support ATE, JATE 2.0 extends Solr's text analysis libraries by supporting three types of linguistic filters for term candidate extraction. These include: (1) a PoS pattern based chunker that extracts candidates based on user specified patterns (e.g., line 9 of schema.xml in Figure 2); (2) a token N-gram extractor that extends the built-in one; (3) a noun phrase chunker. All of these are implemented with the capability to normalize noisy terms, such as removing leading and trailing stop words, and non-alphanumeric tokens. In addition, JATE 2.0 also implements an English lemmatizer, which is a recommended replacement of stemmers that can sometimes be too aggressive for ATE. These offer great flexibility enabling almost any customized term candidate extraction. Figure 2 shows a standard configuration of an text analyzer for PoS pattern based candidate extraction for English.

3.3. Filtering

Extracted term candidates are further processed to make a final decision to separate terms from non-terms.

3.3.1. Pre-filtering

It is often a common practice to apply a filtering process to term candidates to reduce both noise (e.g., due to irregular textual data or erroneous PoS tagging) and computation before the subsequent step of term scoring and ranking. The

[14] `https://cwiki.apache.org/confluence/display/solr/Uploading+Data+with+Solr+Cell+using+Apache+Tika`

[15] `https://tika.apache.org`

[16] `https://wiki.apache.org/solr/ExtractingRequestHandler`

[17] `https://cwiki.apache.org/confluence/display/solr/Detecting+Languages+During+Indexing`

[18] `https://cwiki.apache.org/confluence/display/solr/CharFilterFactories`

[19] `https://cwiki.apache.org/confluence/display/solr/Solr+Glossary`

[20] For practical reasons, we rely on a separate field that indexes the same content as token N-grams and stored term vectors which provides the lookup of statistics about term candidates. Details on this can be found on the JATE website and Solr documentation for Inverted Index.

[21] See Solr API documentation.

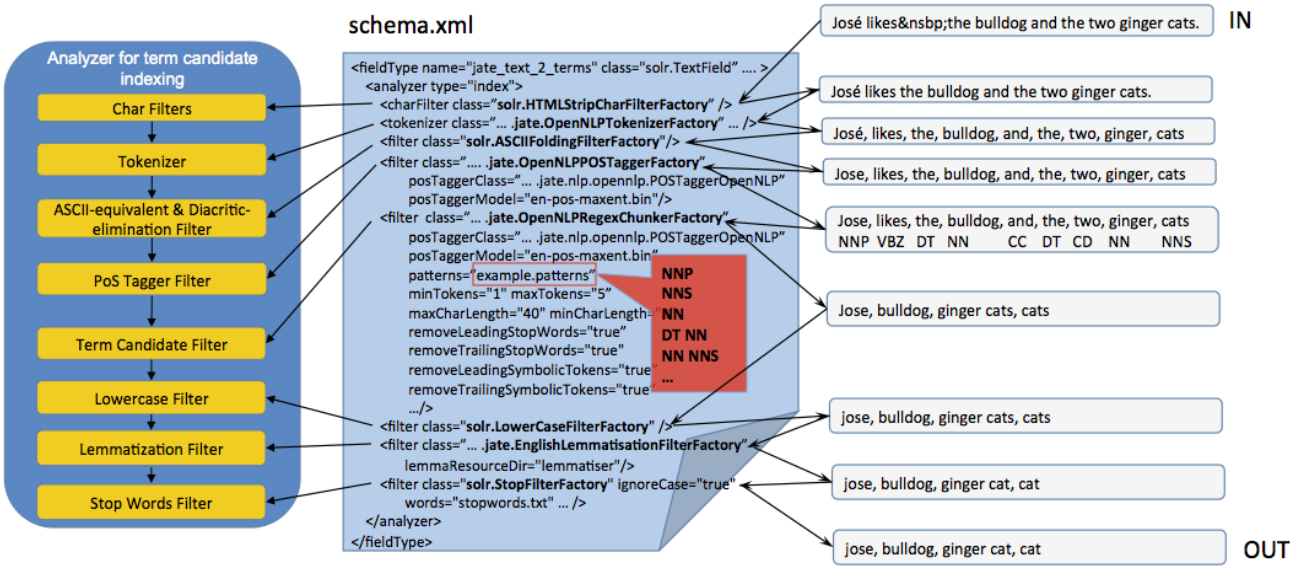

Figure 2: Example of PoS pattern based candidate extraction

following strategies have been implemented in JATE 2.0 and can be configured according to user needs to balance precision and recall.

Minimal and maximal character length restricts candidates to a fixed range of character length. **Minimal and maximal token number** requires a valid candidate to contain certain number of tokens (e.g., to limit multi-word expressions). **Minimal stop words removal** removes leading and/or trailing stop words in a multi-word expression (e.g.,'the cat' becomes 'cat', but 'Tower of London' remains intact). All these are implemented as part of the three term candidate extraction methods described before and can be configured in the analyzer.

Frequency threshold allows candidates to be filtered by a minimum total frequency in the corpus. This is often used in typical ATE methods, and is configurable via individual scoring and ranking algorithms that follow.

3.3.2. Scoring and ranking

In this step, term candidates that pass the pre-filter are scored and ranked. Their basic statistical information is gathered from the Solr index, to create complex features required by different algorithms.

JATE 2.0 has implemented 10 algorithms for scoring candidates, listed in Table 1. *TTF* is the total frequency of a candidate in the corpus. It's usage in ATE was firstly documented in (Justeson and Katz, 1995). *ATTF* divides *TTF* by the number of documents a candidate appears in. *TTF-IDF* adapts the classic document-specific *TF-IDF* used in IR to work at corpus level, by replacing *TF* with *TTF*.

RIDF was initially proposed by (Church and Gale, 1995) as an enhancement to IDF to identify keywords in a document collection. It measures the deviation of the actual IDF score of a word from its 'expected' IDF score, which is predicted based on a Poisson distribution. The hypothesis is based on the fact that Poisson model fits poorly with such keywords. Thus a prediction of IDF based on Poisson can deviate from its actual IDF observed based on a corpus. Empirically, it is shown that all words have real IDF scores that deviate from the expected value under a Poisson distribution. However, keywords tend to have larger deviations than non-keywords. We adapt it to work with term candidates that can be either single words or multi-word expressions.

CValue observes that real terms in technical domains are often long, multi-word expressions and usually not nested in other terms (i.e., as part of the longer terms). Frequency-based methods are not effective for such terms as (1) nested term candidates will have at least the same and often higher frequency, and (2) the fact that a longer string appears n times is a lot more important than that of a shorter string appearing n times. Thus *CValue* computes a score that is based on the frequency of a candidate and its length, then adjusted by the frequency of longer candidates that contain it.

Similarly, *RAKE* is designed to favour longer multi-word expressions. It firstly computes a score for individual words based on two components: one that favours words occurring often and in longer term candidates, and one that favours words occurring frequently regardless of the words which they co-occur with. Then it adds up the scores of composing words for a candidate.

χ^2 promotes term candidates that co-occur very often with 'frequent' candidates in the corpus. First, candidates are ranked by frequency in the corpus and a subset (typically top $n\%$) is selected - to be called 'frequent terms'. Next, candidates are scored based on the degree to which their co-occurrence with these frequent terms are biased. These biases can be due to semantic, lexical, or other relations of two terms. Thus, a candidate showing strong co-occurrence biases with frequent terms may have an important meaning. To evaluate the statistical significance of the biases, χ^2 is used. For each candidate, co-occurrence frequency with the frequent terms is regarded as a sample value. The null hypothesis that we expect to reject is that 'occurrence of a candidate is independent from occurrence of frequent

Table 1: ATE algorithms implemented in JATE 2.0

Name	Full name	Ref.
TTF	Term Total Frequency	(Justeson and Katz, 1995), FiveFilters.org
ATTF	Average TTF	-
TTF-IDF	TTF and Inverse Doc Frequency	-
RIDF	Residual IDF	(Church and Gale, 1995)
CValue	C-Value	(Ananiadou, 1994)
χ^2	Chi-Square	(Matsuo and Ishizuka, 2003)
RAKE	Rapid Keyword Extraction	(Rose et al., 2010)
Weirdness	Weirdness	(Ahmad et al., 1999)
GlossEx	Glossry Extraction	(Park et al., 2002)
TermEx	Term Extraction	(Sclano and Velardi, 2007)

terms'.

Both *RAKE* and χ^2 were initially developed for extracting document-specific keywords. We adapt them for ATE from document collections. This is done by replacing document-level frequency with corpus-level frequency, wherever needed.

Weirdness is a contrastive approach (Drouin, 2003) which is particularly interesting when trying to identify low-frequency terms. The method compares normalized frequency of a term candidate in a domain-specific corpus with a reference corpus, such as the general-purpose British National Corpus[22]. The idea is that candidates appearing more often in the target corpus are more specific to that corpus and therefore, more likely to be real terms. To cope with out-of-vocabulary candidates, we modify this by taking the sum of the *Weirdness* scores of composing words for a candidate.

Both *GlossEx* and *TermEx* extend *Weirdness*. *GlossEx* linearly combines 'domain specificity', which normalizes the *Weirdness* score by the length (number of words) of a term candidate, with 'term cohesion' that measures the degree to which the composing words tend to occur together as a candidate other than appearing individually. This is computed based on comparing the frequency of a candidate against its composing words. *TermEx*, in a very similar form, further extends *GlossEx* by linearly combining a third component that promotes candidates with an even probability distribution across the documents in the corpus (in an analogy, the candidate 'gains consensus' among the documents).

Essentially both *GlossEx* and *TermEx* combine termhood with unithood, which exploits a reference corpus. For these we introduce a scalar to balance the contribution of unithood, which tends to dominate the final score in case of largely disproportionate sizes of the domain and reference corpora (e.g., orders of magnitude difference). The scalar adjusts the normalized frequency of words in both the target and reference corpora to be in the same orders of magnitude[23].

[22] `http://www.natcorp.ox.ac.uk`

[23] Details can be found in the implementation of the two algorithms in JATE 2.0.

3.3.3. Threshold cutoff

Next, a cutoff decision is to be made to separate real terms from non-terms based on their scores. Three options are supported: (1) hard threshold, where term candidates with scores no less than the threshold are chosen; (2) top K, where top ranked K candidates are chosen; (3) and top $K\%$, where the top ranked K percentage of candidates are chosen.

4. Using JATE 2.0 in Two Modes

4.1. Embedded mode

The embedded mode is recommended when users need a list of domain-specific terms from a corpus to be used in subsequent knowledge engineering tasks. Users configure a Solr instance and in particular, a text analysis chain that defines how term candidates are extracted and normalized. The Solr instance is instantiated as an embedded[24] module that interacts with other components. Users then explicitly start an indexing process on a corpus to trigger term candidate extraction, then use specific ATE utility classes[25], each wrapping an individual ATE algorithm, to perform candidate scoring, ranking, filtering and export.

4.2. Plugin mode

The plugin mode is recommended when users need to index new or enrich existing index using extracted terms, which can, e.g., support faceted search. ATE is performed as a Solr plugin. A `SolrRequestHandler` [26] is implemented so that term extraction can be triggered by HTTP request. Users configure their Solr instance in the same way as above, then start the instance as a back-end server. To trigger ATE, users send an HTTP request to the `SolrRequestHandler`, passing parameters specifying the input data and the ATE algorithms. Candidate extraction is optional if the process is performed with document indexing. ATE then begins and when finished, updates documents in the index with extracted terms.

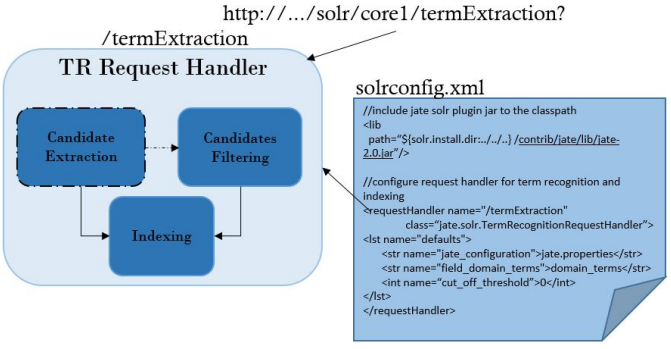

Figure 3: Term Extraction by HTTP request (Plugin mode)

[24] `https://wiki.apache.org/solr/EmbeddedSolr`

[25] See details in the `app` package of JATE 2.0

[26] `https://wiki.apache.org/solr/SolrRequestHandler`

5. Experiment

We evaluate JATE 2.0 on two datasets, the GENIA dataset (Kim et al., 2003), a semantically annotated corpus for biotextmining previously used by (Zhang et al., 2008); and the ACL RD-TEC dataset (Zadeh and Handschuh, 2014), containing publications in the domain of computational linguistics and a list of manually annotated domain-specific terms.

GENIA contains 1,999 Medline abstracts, selected using a PubMed query for the terms 'human', 'blood cells', and 'transcription factors'. The corpus is annotated with various levels of linguistic and semantic information. We extract any text annotated as 'cons' (concept) to compile a gold standard list of terms. ACL RD-TEC contains over 10,900 scientific publications. In (Zadeh and Handschuh, 2014), the corpus is automatically segmented and PoS tagged. Candidate terms are extracted by applying a list of patterns based on PoS sequence. These are then ranked by several ATE algorithms, and the top set of over 82,000 candidates are manually annotated as valid or invalid. We use the valid candidates as gold standard terms. We notice three different versions of corpus available[27]: one only contains the sentences where at least one of the annotated candidate terms must be present (under 'annotation'), one contains complete raw text files in XML format (under 'cleansed_text'), and one contains plain text files from the ACL ARC (under 'external_resource'). We use the XML version and only extract content from '<title>' and '<paragraph>' elements.

We ran all experiments on a server with 16 CPUs. All algorithms follow the same candidate extraction process, unless otherwise noted. Detailed configurations are as follows:

- three different analyzers have been tested, each using PoS sequence pattern based (PoS based) , noun phrase chunking based (NP Chunking based), and the N-gram based term candidate extraction. The pipelines are slightly different for each analyzer and each dataset. Detailed configuration can be found in the example section of the JATE website;
- for PoS-based term candidate extraction, we use: for GENIA, a list of PoS sequence patterns defined in (Ananiadou, 1994); for ACL RD-TEC, a list of PoS sequence patterns defined in (Zadeh and Handschuh, 2014);
- min. character length of 2; max. of 40;
- min. tokens of 1, max. of 5;
- leading and trailing stop words and non-alphanumeric character removal. (for N-gram based removal of *any* non-alphanumeric characters);
- stop words removal;
- min. frequency threshold of 2;
- for χ^2, the 'frequent terms' are selected as top 10%. Moreover, only candidates that appear in at least two sentences are considered;
- for *Weirdness*, *GlossEx* and *TermEx*, the BNC corpus is used as reference;

[27] `http://atmykitchen.info/datasets/acl_rd_tec/`. Accessed: 10 March 2016

Table 2: Comparision of candidate extraction on GENIA (run in a single thread)

Method	Term candidates	Recall	CPU time (millsec)
PoS-based	10,582/38,850	10%	68,914
NP-based	35,799/44,479	23%	118,753
N-gram	48,945/440,974	16%	33,721

Table 3: Comparision of candidate extraction on ACL RD-TEC (run in a single thread)

Method	Term candidates	Recall	CPU time (min)
PoS-based	524,662/1,569,877	74%	133.84
NP-based	585,012/2,014,916	66%	367
N-gram	887,316/7,776,457	41%	189.20

The total number of term candidates extracted for each dataset under each analyzer setting after/before applying the minimum frequency threshold with associated overall recall and CPU time are shown in Tables 2 and 3. The low recall for GENIA may be due to several reasons. First, a substantial part of the gold standard terms are 'irregular', as they contain numbers, punctuations and symbols. The patterns in our experiment are mainly based on adjectives and nouns, and will not match irregular terms. Second, we use a general purpose PoS tagger which does not perform well for the biomedical text. In addition, GENIA consists of very short abstracts and as a result, many legitimate terms may be removed due to the frequency threshold and lexical pruning. However, this can be easily rectified by relaxing the pre-filters.

We then show precision of top *K* terms ranked by each algorithm, commonly used in ATE. Figure 4 shows results for the GENIA dataset and Figure 5 shows results for the ACL RD-TEC dataset. First, all algorithms achieve very high precision on the GENIA dataset. This is partly because the gold standard terms are very densely distributed. Our analysis shows that 49% of words are annotated as part of a term. Second, *TFIDF* and *CValue* appear to be the most reliable methods as they obtain consistently good results on both datasets. Third, *Weirdness*, *GlossEx* and *TermEx* are very sensitive to the choice of reference corpus. For example, on the ACL RD-TEC dataset, phrases containing 'treebank' are very highly ranked. However, many of them are not valid terms. Finally, *RAKE* is the worst performing algorithm on both datasets, possibly because it is very specifically tailored to document-level (other than corpus-level) keyword extraction.

Next we compare the efficiency of each algorithm for scoring and ranking term candidates, by measuring the maximum memory footprint and CPU time for computation. We found consistent patterns among different algorithms, despite what term candidate extractor is used (which only affects absolute figures as it changes the number of candidates generated, see Tables 2 and 3). Using the PoS-based term candidate extractor as example, we show the statistics in Table 4 and Table 5 for the two datasets. As it is shown, χ^2 is the most memory intensive due to the in-memory storage of co-occurrence. This largely depends on the number of term candidates and frequent terms. In terms of speed,

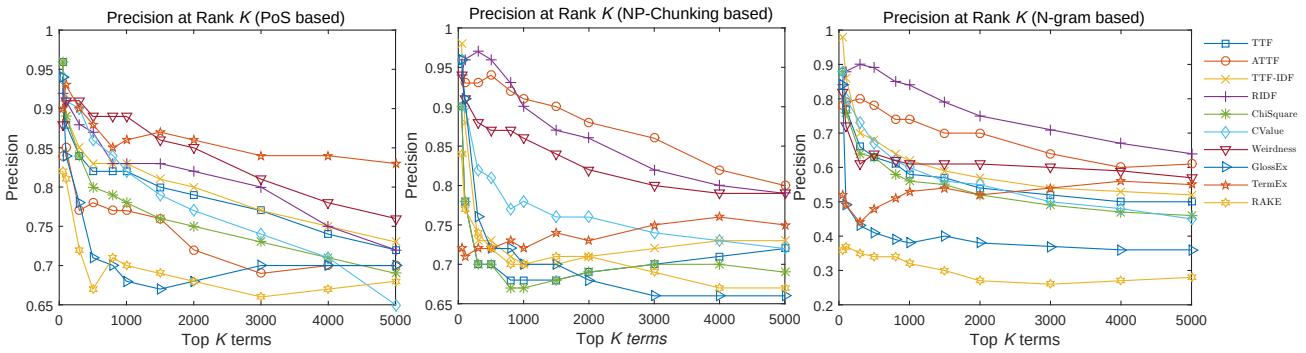

Figure 4: Comparison of Top K precisions on GENIA

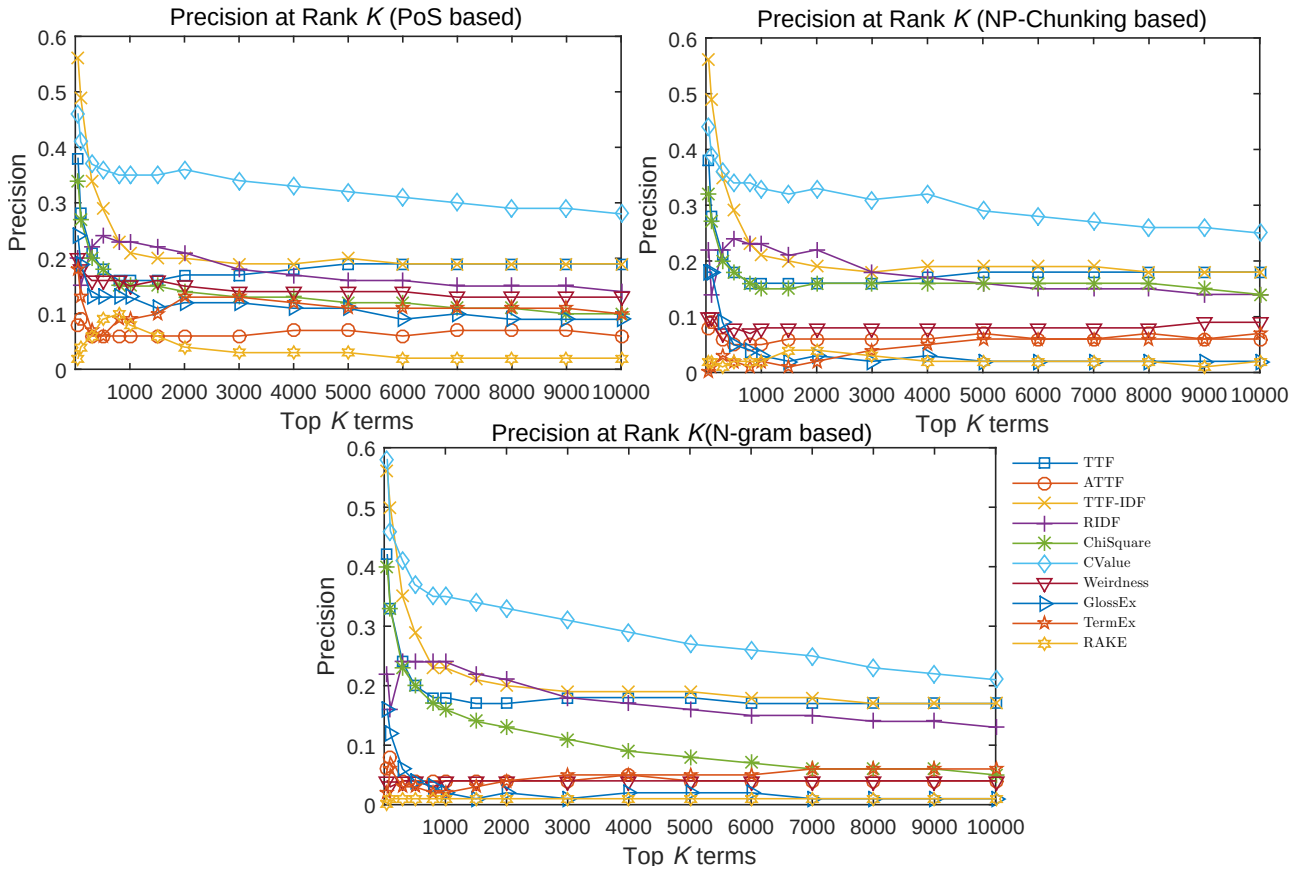

Figure 5: Comparison of Top K precisions on ACL RD-TEC

χ^2, *CValue* and *RAKE* are among the slowest. While χ^2 spends significant time on co-occurrence related computation, *CValue* and *RAKE* spend substantial time on computing the containment relations among candidates.

Table 4: Running time and memory usage on GENIA

Name	CPU time (sec.)	Max vmem (gb)
TTF	19	0.81
ATTF	20	0.83
TTF-IDF	20	0.83
RIDF	20	0.78
χ^2	30	1.12
CValue	47	1.41
Weirdness	22	1.07
GlossEx	25	1.01
TermEx	27	1.06
RAKE	26	1.42

6. Conclusion

This paper describes JATE 2.0, a highly modular, adaptable and scalable ATE library with 10 implemented algorithms, developed within the Apache Solr framework. It advances existing ATE tools mainly by enabling a significant degree of customization and adaptation thanks to the flexibility under the Solr framework; and making available a large collection of state-of-the-art algorithms. It is expected that the tool will bring both academia and industries under a uniform development and benchmarking framework that will encourage collaborative effort in this area of research, to

Table 5: Running time and memory usage on ACL RD-TEC

Name	CPU time (min:sec)	Max vmem (gb)
TTF	5:46	4.2
ATTF	5:56	4.18
TTF-IDF	5:58	4.24
RIDF	5:59	4.38
χ^2	21:47	16.05
CValue	20:59	5.01
Weirdness	7:11	4.89
GlossEx	6:40	4.88
TermEx	9:52	6.12
RAKE	22:59	5.45

foster further contributions in terms of novel algorithms, benchmarkings, support for new languages, new text processing capabilities, and so on. Future work will look into the implementation of additional ATE algorithms, particularly machine learning based methods.

7. Acknowledgement

Part of this research has been sponsored by the EU funded project WeSenseIt under grant agreement number 308429; and the SPEEAK-PC collaboration agreement 101947 of the Innovative UK.

8. Bibliographical References

Ahmad, K., Gillam, L., and Tostevin, L. (1999). University of surrey participation in trec 8: Weirdness indexing for logical document extrapolation and retrieval (wilder). In *Proceedings of the 8th Text REtrieval Conference*.

Ananiadou, S. (1994). A methodology for automatic term recognition. In *Proceedings of the 15th Conference on Computational Linguistics*, pages 1034–1038, Stroudsburg, PA, USA. Association for Computational Linguistics.

Arora, C., Sabetzadeh, M., Briand, L., and Zimmer, F. (2014). Improving requirements glossary construction via clustering: approach and industrial case studies. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, page 18. ACM.

Bowker, L. (2003). Terminology tools for translators. *BENJAMINS TRANSLATION LIBRARY*, 35:49–66.

Brewster, C., Iria, J., Zhang, Z., Ciravegna, F., Guthrie, L., and Wilks, Y. (2007). Dynamic iterative ontology learning. In *Proceedings of the 6th International Conference on Recent Advances in Natural Language Processing*, Borovets, Bulgaria, September.

Church, K. W. and Gale, W. A. (1995). Inverse document frequency (idf): A measure of deviations from poisson. In *Proceedings of the ACL 3rd Workshop on Very Large Corpora*, pages 121–130, Stroudsburg, PA, USA. Association for Computational Linguistics.

Drouin, P. (2003). Term extraction using non-technical corpora as a point of leverage. *Terminology*, 9(1):99–115.

Frantzi, K. T., Ananiadou, S., and Mima, H. (2000). Automatic recognition of multi-word terms:. the c-value/nc-value method. *Natural Language Processing For Digital Libraries*, 3(2):115–130.

Justeson, J. and Katz, S. M. (1995). Technical terminology: some linguistic properties and an algorithm for identification in text. *Natural Language Engineering*, 1(1):9–27.

Kim, J.-D., Ohta, T., Tateisi, Y., and ichi Tsujii, J. (2003). GENIA corpus - a semantically annotated corpus for bio-textmining. In *ISMB (Supplement of Bioinformatics)*, pages 180–182.

labs.translated.net. (2015). Terminology extraction. https://developer.yahoo.com/contentanalysis/ [Online; accessed 9-October-2015].

Matsuo, Y. and Ishizuka, M. (2003). Keyword extraction from a single document using word co-occurrence statistical information. *International Journal on Artificial Intelligence Tools*, 13(1):157–169.

Maynard, D., Saggion, H., Yankova, M., Bontcheva, K., and Peters, W. (2007). Natural language technology for information integration in business intelligence. In *Business Information Systems*, pages 366–380. Springer.

Maynard, D., Li, Y., and Peters, W. (2008). Nlp techniques for term extraction and ontology population. In *Proceedings of the 2008 Conference on Ontology Learning and Population: Bridging the Gap Between Text and Knowledge*, pages 107–127, Amsterdam, The Netherlands, The Netherlands. IOS Press.

Park, Y., Byrd, R. J., and Boguraev, B. K. (2002). Automatic glossary extraction: Beyond terminology identification. In *Proceedings of the 19th International Conference on Computational Linguistics*, pages 1–7, Stroudsburg, PA, USA. Association for Computational Linguistics.

Rose, S., Engel, D., Cramer, N., and Cowley, W., (2010). *Automatic keyword extraction from individual documents*. John Wiley and Sons.

Sclano, F. and Velardi, P. (2007). Termextractor: a web application to learn the shared terminology of emergent web communities. In *Proceedings of the 3rd International Conference on Interoperability for Enterprise Software and Applications*.

Spasić, I., Greenwood, M., Preece, A., Francis, N., and Elwyn, G. (2013). Flexiterm: a flexible term recognition method. *Journal of Biomedical Semantics*, 4(27).

Yahoo! (2015). Yahoo! content analysis api. https://developer.yahoo.com/contentanalysis/ [Online; accessed 9-October-2015].

Zadeh, B. and Handschuh, S. (2014). The acl rd-tec: A dataset for benchmarking terminology extraction and classification in computational linguistics. In *Proceedings of the 4th International Workshop on Computational Terminology (Computerm)*, pages 52–63, Dublin, Ireland, August. Association for Computational Linguistics and Dublin City University.

Zhang, Z., Iria, J., Brewster, C., and Ciravegna, F. (2008). A comparative evaluation of term recognition algorithms. In *Proceedings of The 6th international conference on Language Resources and Evaluation*, Marrakech, Morocco, May.