ATR4S: toolkit with state-of-the-art automatic terms recognition methods in Scala

Nikita Astrakhantsev

Published online: 21 December 2017
© Springer Science+Business Media B.V., part of Springer Nature 2017

Abstract Automatically recognized terminology is widely used for various domain-specific texts processing tasks, such as machine translation, information retrieval or ontology construction. However, there is still no agreement on which methods are best suited for particular settings and, moreover, there is no reliable comparison of already developed methods. We believe that one of the main reasons is the lack of state-of-the-art method implementations, which are usually non-trivial to recreate—mostly, in terms of software engineering efforts. In order to address these issues, we present ATR4S, an open-source software written in Scala that comprises 13 state-of-the-art methods for automatic terminology recognition (ATR) and implements the whole pipeline from text document preprocessing, to term candidates collection, term candidate scoring, and finally, term candidate ranking. It is highly scalable, modular and configurable tool with support of automatic caching. We also compare 13 state-of-the-art methods on 7 open datasets by average precision and processing time. Experimental comparison reveals that no single method demonstrates best average precision for all datasets and that other available tools for ATR do not contain the best methods.

Keywords Automatic term recognition · Terminology extraction · Open source software

The reported study was supported by RFBR, research project No. 14-07-00692.

Nikita Astrakhantsev
astrakhantsev@ispras.ru

1 Ivannikov Institute for System Programming of the Russian Academy of Sciences (ISP RAS), ul. Solzhenitsyna 25, Moscow, Russia 109004
1 Introduction

Automatic terminology recognition (ATR) aims at extraction of terms—words and collocations designating domain-specific concepts—from a collection of text documents belonging to that domain. Extracted terms then can be used for many tasks including machine translation (Gaussier 1998), information retrieval (Lingpeng et al. 2005), sentiment analysis (Mayorov et al. 2015), ontology construction and ontology enrichment (Astrakhantsev et al. 2015).

Despite this importance, the ATR task is still far from being solved: researches continue to propose new methods for ATR, which usually show average precision below 80% even on top 500–1000 terms (Fedorenko et al. 2013; Zhang et al. 2008, 2016) and thus are hardly used in practice. Moreover, there is still no fair and reliable comparison of already developed methods. Most works compare 1–2 newly proposed methods with several old baselines on 1–3 datasets only.

The reason is that a lot of effort is required both to obtain datasets for comparison and to reimplement methods, which are usually non-trivial and dependent on proprietary modules.

To address these issues, we present an open-source implementation of ATR methods and their comparison on many datasets. In more detail, our main contributions are the following:

1. ATR4S1: an open-source implementation of 13 state-of-the-art methods for ATR written in Scala.2
2. Modification of KeyConceptRelatedness method (Astrakhantsev 2015) by replacing proprietary module for semantic relatedness computation with an open-source tool word2vec (Mikolov et al. 2013).
3. Comparison of 13 state-of-the-art methods on 7 datasets by average precision and processing time. In particular, more correct evaluation of unsupervised methods with many parameters (namely, PU-ATR (Astrakhantsev 2015) and aforementioned KeyConceptRelatedness) by adapting the cross-validation strategy.

This paper is organized as follows. Section 2 discuss ATR methods and overview existing software tools for ATR. Section 3 describes ATR4S architecture and implemented methods, including proposed modifications of KeyConceptRelatedness. Section 4 presents experimental evaluation. The last section outlines the paper.

1 https://github.com/ispras/atr4s.
2 We have chosen Scala for the implementation, because it provides powerful toolset for parallelizable processing of big collections, which is especially relevant in case of pipeline architectures of state-of-the-art ATR methods, see Sect. 3. Also note that Scala programs can be easily used in any programming language working on Java Virtual Machine including Java itself; moreover, ATR4S includes for each facade class such interfaces that do not contain any non-Java class.
2 Related work

2.1 ATR methods

The first survey by Kageura and Umino (1996) devoted to ATR distinguished all methods into linguistic and statistical. Then, in a survey by Pazienza et al. (2005) of 2005, it was argued that all modern algorithms include linguistic methods as a filtering step. Finally, the most recent survey by Astrakhantsev et al. (2015) identified the general pipeline of ATR methods: preprocessing, term candidates collection, term candidate scoring, and term candidate ranking.

Preprocessing transforms input text into a sequence of elements needed for further term candidate extraction; most often, each of such elements consists of lemmatized token with attached part of speech tag; some works (Judea et al. 2014; Zhang et al. 2016) instead use noun phrases obtained by shallow parsing.

Term candidate extraction can be seen as a filtering step: it should throw out such words and collocations that are almost certainly not terms, based on simple linguistic and statistical criteria like presence of stop word or minimal frequency of occurrence.

Term candidate scoring, i.e. assigning a number to each term candidate reflecting its likelihood of being a term, is the most important and sophisticated step in the whole pipeline. Considering the type of information used to score term candidate, we can form the following groups: methods based on frequencies of term candidate occurrences (with large subgroup of word association measures); on occurrences contexts; on reference corpora; on topic modeling; on Wikipedia. ATR4S includes most promising methods for each group; see details in Sect. 3.3.

Term candidate ranking is the final step, which lets to take the top candidates and thus distinguish terms from not terms. It is trivial in the case of only one term scoring method, because we can simply rank by that score; and the vast majority of works belongs to this group. Some works use linear combination, voting algorithm or semi-supervised learning, we discuss them in Sect. 3.4. Another set of works (Astrakhantsev et al. 2014; Fedorenko et al. 2013; Nokel and Loukachevitch 2013) apply supervised machine learning.

Note that the pipeline outlined above does not cover all possible ATR methods, for example those based on rules or sophisticated linguistic knowledge, but in order to simplify architecture, we do not include such methods into the first version of ATR4S.

There are also several works focusing on experimental evaluation of ATR (Wermter and Hahn 2006; Zhang et al. 2008; Mondary et al. 2012; Fedorenko et al. 2013; Nokel and Loukachevitch 2013), but, as mentioned above, numbers of datasets and methods used in such experiments are too low for making reliable conclusions.
2.2 ATR software tools

Many software tools have been developed for ATR to date. However, most of them provide only 1 or 2 methods, which are usually outdated. For example, TerMine\(^3\) is based on CValue/NC-Value methods (and academic usage only); FlexiTerm contains C-Value and “a simple term variant normalisation method” (Spasić et al. 2013); TOPIA\(^4\) lists only one method without algorithm description and it is not updated since 2009; TermRider\(^5\) utilizes TF-IDF only; TermSuite (Cram and Daille 2016) ranks candidates by Weirdness method, but focuses on recognizing term variants based on syntactic and morphological patterns.

Some tools are limited by searching for mentions of (named) entities (for example, OpenCalais\(^6\)) or named entities and Wikipedia concepts [Texterra (Turdakov et al. 2014)]. Another tool\(^7\) supports only supervised recognition of 1-word and 2-words terms.

JATE 2.0 (Zhang et al. 2016) is the most similar tool to ATR4S: it is written in Java and also can be natively used in any JVM-based language, contains many ATR algorithms and multiple methods for term candidates collection; it is highly modular and adaptable. However, it lacks a lot of actual state-of-the-art methods, namely those based on occurrences contexts, topic models, Wikipedia, and non-trivial ranking algorithms such as Voting (Zhang et al. 2008) and PU-ATR (Astrakhantsev 2015). It also depends on Apache Solr,\(^8\) which may simplify its integration to the application that already uses Solr, but may as well complicate its usage as a library.

3 Architecture

ATR4S follows general pipeline of term recognition: texts preprocessing, term candidates collection, term candidate scoring, and term candidate ranking. Subsections below describe each step in details.

3.1 Preprocessing

At the preprocessing step, ATR4S splits input text documents into sentences, tokenizes obtained sentences, and finds part of speech tags and lemmas for obtained tokens. In order to perform these tasks, ATR4S incorporated 3 external NLP

\(^{3}\) http://www.nactem.ac.uk/software/termine.
\(^{4}\) https://pypi.python.org/pypi/topia.termextract.
\(^{5}\) https://gate.ac.uk/projects/neon/termraider.html.
\(^{6}\) http://www.opencalais.com/about-open-calais/.
\(^{7}\) https://bitbucket.org/Meister17/term-extraction.
\(^{8}\) http://lucene.apache.org/solr/.
libraries: Stanford CoreNLP (Manning et al. 2014), Emory nlp4j\(^9\) and Apache OpenNLP.\(^{10}\) We use the first one in all experiments.\(^{11}\)

### 3.2 Term candidates collection

ATR4S extracts consecutive word n-grams of specified orders (by default, from 1 to 4) as term candidates. Three basic filters can be applied before formation of term candidate occurrence (or term occurrence, for brevity):

1. Noise word filter: keeps term occurrence if all of its lemmas have length not less than the predefined limit and match the predefined regular expression (by default, length limit is 3 characters and the regular expression filters out words containing non-alphanumeric characters). This filter is most useful for texts obtained from automatic parsing (e.g. PDF or HTML) and thus containing a lot of noise words.
2. Stop word filter: keeps term occurrence if the predefined set of stop words contains no lemma of the term occurrence. By default, we use stop words list from the SMART retrieval system Salton (1971).
3. Part of speech (PoS) tags pattern\(^{12}\): keeps term occurrence if its PoS tags match the pattern encoded as the regular expression. By default, we apply the commonly-used pattern (Buitelaar and Eigner 2009) extended by allowing prepositions between nouns: \((\text{NN}(S)?|\text{JJ}|\text{NNP}|\text{NN}(S?)\text{IN})*\text{NN}(S)?)\)

Then ATR4S combines occurrences with the same canonical representation (lemmas joined by underscore symbol, e.g. `information_processing`) as belonging to the same term candidate.

Finally, ATR4S filters out term candidates occurring rarer than the predefined number of times (by default, 2), in order to reduce both computation efforts and noise occurring due to errors in preprocessing steps or input data preparation.

### 3.3 Term candidate scoring

ATR4S includes 13 methods for term candidate scoring; below we describe them grouped by the type of information used to score term candidate.

---

\(^9\) https://emorynlp.github.io/nlp4j/.

\(^{10}\) http://opennlp.apache.org/.

\(^{11}\) Preliminary experiments showed drop of 1–5% in average precision in the case of switching from CoreNLP, mainly due to part of speech tagging errors; however, note that CoreNLP is distributed under GPL, while others are licensed under the Apache License, Version 2.0.

\(^{12}\) Some works use shallow parsing (noun phrase chunking), but our preliminary experiments showed that this works not better, while is more computationally expensive.
3.3.1 Methods based on occurrences frequencies

Most methods for term candidate scoring are based on the intuition that the more frequently some word or collocation occur in the domain-specific text collection, the more likely it is a term for this domain. This subsection describes methods that utilize this intuition only, i.e. those considering only frequencies of words constituting term candidate and ignoring other information.

Besides Term frequency (TF) itself, this group contains Average term frequency (ATF) (Zhang et al. 2016), TF-IDF (Evans and Lefferts 1995), Residual IDF (RIDF) (Zhang et al. 2016), CValue (Frantzi et al. 2000), Basic (Bordea et al. 2013), and ComboBasic (Astrakhantsev 2015).

ATF simply normalizes term frequency by number of documents containing this term candidate.

TF-IDF is a classical information retrieval measure showing high values for term candidates that occur frequently in few documents:

$$TF \cdot IDF(t) = TF(t) \cdot \log \frac{D}{DTF(t)},$$

where $D$ is a total number of document in collection, $DTF(t)$ is a number of documents containing term candidate $t$.

RIDF was originally proposed for keywords extraction (Church and Gale 1999) and than re-used for term recognition (Zhang et al. 2016). It is based on the assumption that the deviation of observed IDF from the IDF modeled by Poisson distribution is higher for keywords than for ordinary words.

$$RIDF(t) = TF(t) \cdot \log \frac{D}{DTF(t)} + \log(1 - e^{-ATF(t)}),$$

C-Value, one of the most popular methods, promotes term candidates that occur frequently, but not as parts of other term candidates. This method was supposed to work with multi-word term candidates only; ATR4S includes modification proposed by Ventura et al. (2013) that supports one-word term candidates as well:

$$C-Value(t) = \begin{cases} 
\log_2(|t| + \alpha) \cdot TF(t), & \text{if } \{s : t \subseteq s\} = \emptyset; \\
\log_2(|t| + \alpha) \cdot \left( TF(t) - \sum_{s : t \subseteq s} TF(s) \right), & \text{else}. 
\end{cases}$$

where $|t|$ is a length of term candidate $t$ (number of words), $s$ is a set of term candidates containing $t$, i.e. such candidates that $t$ is a their substring, $\alpha$ is a smoothing parameter for one-word terms; Ventura et al. suggested to use $\alpha = 1$, but during preliminary experiments we found that $\alpha = 0.1$ leads to better results; therefore ATR4S uses $\alpha = 0.1$ by default.

Basic is a modification of C-Value for intermediate level (of specificity) terms extraction. Like original C-Value, it can extract multi-word terms only; however, unlike C-Value, Basic promotes term candidates that are part of other term candidates.
candidates, because such terms are usually served for creation of more specific terms.

\[ \text{Basic}(t) = |t| \log f(t) + xe_t, \]  

(4)

where \( e_t \) is a number of term candidates containing \( t \).

ComboBasic modifies Basic further, so that the level of term specificity can be customized by changing parameters of the method:

\[ \text{ComboBasic}(t) = |t| \log f(t) + xe_t + \beta e_t', \]  

(5)

where \( e_t' \) is a number of term candidates that are contained in \( t \). Therefore, by increasing \( \beta \), one can extract more specific terms and vice versa.

Note that ATR4S does not include methods based on word association measures like Lexical Cohesion (Park et al. 2002), Term Cohesion (Kozakov et al. 2004) or classical information-theoretic and statistical measures (z-test, t-test, \( \chi^2 \)-test, loglikelihood, mutual information), because they were repeatedly shown to obtain not better results than simple frequency (Wermter and Hahn 2006; Nokel and Loukachevitch 2013; Zhang et al. 2016).

### 3.3.2 Methods based on occurrences contexts

Methods from this group follow the distributional hypothesis (Harris 1954) and try to distinguish terms from non-terms by considering their contexts. We are aware of only 2 such methods: NC-Value (Frantzi et al. 2000) and DomainCoherence (Bordea et al. 2013); since the latter is a modification of the former and was shown to work better (Bordea et al. 2013), ATR4S includes only it.

DomainCoherence works in 3 steps. First, it extracts the 200 best term candidates by using Basic method.

Then, words from contexts of previously extracted 200 terms are filtered: it keeps only nouns, adjectives, verbs and adverbs that occur in at least one quarter of all documents and are similar to these 200 term candidates, i.e. ranked in the top 50 by averaged Normalized PMI:

\[ s(w) = \frac{1}{|T|} \sum_{t \in T} \text{NPMI}(t, w) = \frac{1}{|T|} \sum_{t \in T} \log \left( \frac{P(t, w)}{P(t)P(w)} \right), \]  

(6)

where \( w \) is a context word; \( T \) is a set of the 200 best term candidates extracted by Basic; \( P(t, w) \) is a probability of occurrence of word \( w \) in the context of \( t \); \( P(t) \) and \( P(w) \) are probabilities of occurrences of term \( t \) and word \( w \), correspondingly. These probabilities are estimated on the basis of occurrence frequencies in the input collection; context is considered to be a 5 words window.

Finally, as a weight of a term candidate, DomainCoherence takes the average of the same NPMI measures computed with each of 50 context words extracted at the previous step.
3.3.3 Methods based on reference corpora

There are multiple methods based on the assumption that terms can be distinguished from other words and collocations by comparing occurrences statistics of considered domain-specific collection with statistics of some reference corpus (usually, from general domain).

DomainPertinence (Meijer et al. 2014) is the simplest implementation of this idea:

\[
\text{DomainPertinence}(t) = \frac{\text{TF}_{\text{target}}(t)}{\text{TF}_{\text{reference}}(t)},
\]

(7)

where \( \text{TF}_{\text{target}}(t) \) is a frequency of term candidate \( t \) in target (domain-specific) collection; \( \text{TF}_{\text{reference}} \) is a frequency in reference (general) collection.

Weirdness (Ahmad et al. 1999) normalizes it by sizes (in number of words) of document collections:

\[
\text{Weirdness}(t) = \frac{\text{NTF}_{\text{target}}(t)}{\text{NTF}_{\text{reference}}(t)},
\]

(8)

where \( \text{NTF}_{\text{target}}(t) \) and \( \text{NTF}_{\text{reference}} \) are frequencies of \( t \) normalized by sizes of target and reference collections, respectively.\(^{13}\)

Relevance (Peñas et al. 2001) further updates it by taking into account fraction of documents, where term candidate occur:

\[
\text{Relevance}(t) = 1 - \left( \log_2 \left( 2 + \frac{\text{NTF}_{\text{target}}(t) \cdot \text{DF}_{\text{target}}(t)}{\text{NTF}_{\text{reference}}(t)} \right) \right)^{-1}
\]

(9)

ATR4S uses Corpus of Historical American English\(^{14}\) as a reference collection.

3.3.4 Methods based on topic modeling

These methods are based on the idea that topic modeling uncovers semantic information useful for term recognition; in particular, that distribution of words over topics found by topic modeling is a less noisy signal than simple frequency of occurrences.

To the best of our knowledge, this group contains only one method capable to extract terms of arbitrary length, that is Novel Topic Model (Li et al. 2013).

First, it obtains probability distribution of words over the following topics: \( \phi^t \)—general topics (\( 1 \leq t \leq 20 \)); \( \phi^B \)—background topic; \( \phi^D \)—document-specific topic. Then, it extracts 200 words most probable for each topic: \( V_t, V_B, V_D \),

\(^{13}\) Note that in the case of simple ranking Weirdness and DomainPertinence show exact the same results, because scores for the same term candidate computed by these 2 methods differ by a constant multiplier only.

\(^{14}\) http://www.ngrams.info/download_coha.asp.
correspondingly; finally, for each term candidate $c_i$ its weight is computed as a sum of maximal probabilities for each of its $L_i$ words ($w_{i1}w_{i2}\cdots w_{iL_i}$):

$$NTM(c_i) = \log(TF_i) \cdot \sum_{1 \leq j \leq L_i, w_j \in \{V_t\}_{t \in \mathcal{T} \setminus \{B,D\}}} \phi_{wj}^{mt_{wj}},$$

(10)

where $mt_{wj} = \star \arg \max_{t \in \mathcal{T} \setminus \{B,D\}} \phi_{wj}^t$.

For topic modeling, ATR4S uses an open source framework. 15

3.3.5 Methods based on Wikipedia

All methods mentioned above require large collection of text documents, otherwise information about term candidate occurrences is too noisy. The only way to overcome this in the case of a small collection is to use external resources. Such a resource should satisfy two requirements: (a) it should be specific enough to contain domain-specific information needed to distinguish terms from not terms; (b) it should be general enough to be applicable for many domains in practice. Wikipedia 16 satisfies these requirements: it is multilingual (English version contains more than 5 million articles), covers a lot of domains and keeps growing.

One of the simplest methods in this group is WikiPresence: it returns 1 if term candidate occurs in Wikipedia pages as hyperlink caption; 0 otherwise. Example of its usage is an additional filter for other methods (Astrakhantsev 2015). LinkProbability (Astrakhantsev 2014) is a normalized frequency of being a hyperlink in Wikipedia pages:

$$LinkProb_{T}(t) = \begin{cases} 0, & \text{if Wikipedia does not contain } t \text{ or } \frac{H(t)}{W(t)} < T; \\ \frac{H(t)}{W(t)}, & \text{else.} \end{cases}$$

(11)

Here $H(t)$ is a number of occurrences of term candidate $t$ as a hyperlink caption; $W(t)$ is a total number of occurrences in Wikipedia pages; $T$ is a method parameter needed to filter out too small values, because they occur due to markup errors in most cases; experimentally chosen value $T = 0.018$ is used by default.

This method propagates term candidates that are specific enough to be provided with a hyperlink; however, it is able to distinguish terms from general words and collocations, but not from terms of other domains.

KeyConceptsRelatedness (Astrakhantsev 2014) interprets domain-specific terms as words and collocations that are semantically related to knowingly domain-specific concepts. This method assumes concepts that are key for many documents in the input collection to be a good approximation for such knowingly domain-specific concepts.

Originally, it was based on computation of semantic relatedness between two Wikipedia concepts (i.e. pages) by Dice measure, which is a count of common

15 https://github.com/ispras/tm.
16 https://www.wikipedia.org/.
neighbors of these pages divided by total neighbors. We modified this: instead of Dice measure, we use cosine distance between word embedding vectors (Mikolov et al. 2013) corresponding to Wikipedia concepts. More precisely, we preprocess the Wikipedia dump by removing markup, while keeping occurrences of Wikipedia concepts (replace each hyperlink by special token that includes title of link’s target concept), and by tokenizing and stemming, then we build a word embedding model and, finally, use it for semantic relatedness computation.

Another modification relates to the extraction of key concepts from a text document. Initially, KeyConceptsRelatedness used an algorithm based on semantic graph construction and clustering, but it works too slowly: in particular, because it requires full word sense disambiguation of all texts.

We propose to use a simplified version of KP-Miner (El-Beltagy and Rafea 2010): in order to be considered as a candidate to key concept, a word or a collocation must: (a) occur at least twice; (b) have an occurrence among the first 800 words; (c) be a valid term candidate, i.e. satisfy requirements listed in Sect. 3.2; and (d) be contained in the vocabulary of constructed word embedding model (i.e. Wikipedia dump should contain at least 5 hyperlinks to the concept with the same title as the word/collocation). Then we rank such candidates by the product of their length (in words) and number of occurrences in the document.

In summary, the modified algorithm for KeyConceptsRelatedness is the following:

1. Extract key concepts for the whole document collection:
   (a) extract $d$ key concepts from each document (see the algorithm above);
   (b) keep $N$ key concepts with maximal frequency (number of being chosen as a key concept).

2. For each term candidate: if the word embedding model does not contain the term candidate, then return 0; otherwise compute semantic relatedness to extracted $N$ key concepts by weighted kNN adapted for the case with only positive instances:

$$sim_k(c, C_N) = \frac{1}{k} \sum_{i=1}^{k} \cos(v_c, v_i)$$

where $c$ is a term candidate; $C_N$ is a set of $N$ key concepts sorted by semantic relatedness to $c$ in descending order; $k$ is a parameter from kNN (should be much smaller than $N$); $v_c$ is an embedding vector corresponding to the term candidate; $v_i$ is an embedding vector corresponding to the key concept $i$.

This method propagates a term candidate that has corresponding Wikipedia article, which is semantically related to key concepts of the whole document.

---

17 https://github.com/phdowling/wiki2vec.
18 https://github.com/RaRe-Technologies/gensim.
19 We keep original constant from KP-Miner algorithm.
collection. Note that this means relying on encyclopedic terms, therefore KeyConceptsRelatedness may be not well suited for those domains having low coverage by Wikipedia.

There are few other methods utilizing Wikipedia. Wu et al. (2012) extract terms from Wikipedia only, not from domain-specific text collection, and require manually chosen categories of the target domain as input. Vivaldi and Rodriguez (2010, 2011); Vivaldi et al. (2012) also take pre-specified categories as input, but work with text collection as well. However, their method was shown (Astrakhantsev 2015) to have worse results than KeyConceptRelatedness (and most other existing methods), while require a lot of engineering and computational efforts, therefore we do not implement this method for ATR4S.

### 3.4 Term candidate ranking

As we already mentioned in Sect. 2, term candidate ranking becomes non-trivial in the case of multiple methods for term candidate scoring. (Following terminology of machine learning, we will refer such methods for scoring as features, for brevity.) General idea for this problem is to aggregate values of multiple features into one number (usually, between 0 and 1), thus reducing the task to ranking by one method.

One of the most popular method is a linear combination of features with some predefined (usually, equal) coefficients. Examples include PostRankDC (Bordea et al. 2013) and GlossEx (Park et al. 2002).

Note that a linear combination does not require scores of other term candidates to be computed in advance, so it is simpler and faster, but misses potentially useful information. The voting algorithm (Zhang et al. 2008) considers values of all term candidates and it was shown (Zhang et al. 2008) to outperform single methods and weighted average (i.e. a linear combination):

\[
V(t) = \sum_{i=1}^{n} \frac{1}{r(f_i(t))},
\]

where \( r(f_i(t)) \) is a rank of term candidate \( t \) among all candidates sorted by feature \( f_i \) only; \( n \) is a total number of aggregated features.

More sophisticated approach is PU-ATR (Astrakhantsev 2014), which is based on the ideas of bootstrapping (like NC-Value and DomainCoherence) and positive unlabeled (PU) learning.

It extracts top 50–200 terms by a single method (seed method); then computes values for multiple features for all term candidates; learns positive-unlabeled classifier by considering these seed terms as positive instances and all other term candidates as unlabeled instances, where each instance is a vector of feature values; and, finally, applies learned classifier to each term candidate, so that the obtained classifier’s confidence is a final aggregated value.

ComboBasic is recommended (Astrakhantsev 2015) as a seed method, because (a) it allows adjusting the level of specificity of seed terms and thus indirectly

---

20 In particular, because it can be easily parallelized by term candidates.
affects the level of specificity of all terms; (b) it is simple enough to include terms of different nature, i.e. terms that can be extracted by using different types of information (see Sect. 3.3), so that PU algorithm overfits less probably.

Note that we perform probabilistic classification, which can suffer from the problem of multicollinearity. Thus, following the previous work (Astrakhantsev 2015), we assume that scoring methods from different groups weakly correlate and choose them as features: C-Value (occurrences frequencies), DomainCoherence (occurrences contexts), Relevance (reference corpora), NovelTopicModel (topic modeling), LinkProbability (Wikipedia, domain-independent specificity), KeyConceptRelatedness (Wikipedia, domain-specificity).

Different Positive-Unlabeled algorithms were shown (Astrakhantsev 2015) to work similarly for this task, so we chose the simplest one (Liu et al. 2002), with Logistic Regression as an internal probabilistic classifier (during preliminary experiments we found it outperforming Random Forest classifier).

3.5 Tool features

ATR4S is highly scalable (by CPUs of one machine), modular and configurable tool that supports automatic caching.

Scalability is provided by storing documents and candidates in Scala parallel collection: preprocessing and most steps of candidates collection can be parallelized by documents; candidates scoring can be parallelized by candidates themselves.

The whole pipeline is instantiated by its own configuration class, which contains corresponding configurations for each constituent step, which are configurable in the same way, i.e. by constructor injection, until the final configurations with constants only, not instances of other configuration classes. This configuration can be serialized/deserialized to/from a human-readable JSON file that can be manually edited, so the tool can be easily configured without a necessity to rebuild it.

The described architecture enables automatic caching: since configuration for each step uniquely determines result of such step, we can cache that result and address it by the corresponding configuration. Considering the observation that ATR methods usually require fine-tuning for optimal quality and thus are often launched many times, such caching can significantly speed up unchanged (previous) steps, e.g. dataset preprocessing or candidates collection, and therefore, speed up the whole process.

4 Evaluation

4.1 Experiments design

We evaluate ATR4S on 7 datasets: GENIA (Kim et al. 2003), FAO (Medelyan and Witten 2008), Krapivin et al. (2009), Patents (Judea et al. 2014), ACL RD-TEC

21 We use Apache Spark MLlib for supervised ML: http://spark.apache.org/mllib/.
22 See details in the source code, class ru.ispras.atr.utils.Cacher.
23 Downloadable here: https://at.ispras.ru/owncloud/index.php/s/kXqCBSryRswThTy.
Note that only 3 datasets (GENIA, Patents, ACL 2.0) provide manual markup of all term occurrences that can be used for reliable evaluation of ATR methods. Expected terms for ACL dataset was formed by manual filtering of top term candidates found by several simple ATR methods, so its recall is obviously less than 100% and some correlation with these simple ATR methods might be introduced. Two other datasets (Krapivin and FAO) were originally used for keyphrase extraction; their expected terms are actually a union of keyphrases of all documents, which is clearly an approximation of true terms. Texts and expected terms of Europarl dataset cover not only politics, but multiple other (mostly related) domains.

We extract term candidates by using default parameters and filters described in Sect. 3.2; Table 2 shows summary statistics of collected candidates.

We use a standard metric for ATR, average precision at level K (AvP):

$$\text{AvP}(K) = \frac{1}{K} \sum_{i=1}^{K} P(i)(R(i) - R(i-1)),$$

where $P(i)$ is precision at level $i$; $R(i)$ is recall at level $i$.

We choose $K$ to be equal to the number of expected terms among extracted term candidates for the dataset (see the last column in Table 2): bigger values of $K$ lead to AvP numbers lesser than 100% even for the perfect algorithm for term candidate ranking, while smaller values of $K$ reduce sensitivity of the quality metric.

In order to find best parameters of the methods modified in this work, that are KeyConceptRelatedness and PU-method, we adapt cross-validation strategy in the following way: each dataset is considered to be a fold; one fold is test—we use it for

---

24 http://eurovoc.europa.eu/drupal.
computing average precision of the parameter set chosen as the best one; other folds are validation—we use them for choosing the best parameter set as follows:

1. evaluate each parameter set on each dataset, i.e. associate each parameter set with the \( \text{AvP} \) obtained on this dataset;
2. find the maximum \( \text{AvP} \) on that dataset;
3. compute relative goodness of each parameter set for each dataset by dividing \( \text{AvP} \) of this parameter set on the maximum \( \text{AvP} \) for this dataset;
4. choose the best (CV) parameter set as the one with maximum product of relative goodesses over all (validation) datasets.

We prefer this strategy to the commonly used optimization over all datasets, because it is more similar to the real setting, when we apply ATR to the new dataset without any labeled data and have to use (or at least start from) parameters that were optimized on some previous datasets. It is especially relevant for the methods with many parameters like KeyConceptRelatedness and PU-method due to their potentially higher chances of overfitting. For the same reason, we also keep one dataset, Europarl, only for the final comparison.

### 4.2 Quality results (average precision)

To optimize parameters of KeyConceptRelatedness method we perform grid search with the following set of possible values: count of key concepts per document \( d = \{3, 5, 10, 15, 20, 30\} \); total count of key concepts \( N = \{50, 100, 200, 300, 500\} \); count of nearest keys \( k = \{1, 2, 3, 5, 10\} \).

As we can see from Table 3, parameters found by cross-validation are quite stable: parameter set \( d = 15, N = 500, k = 2 \) shows the highest result in 4 of 5 cases and in the same 4 cases difference between test \( \text{AvP} \) and best \( \text{AvP} \) is about 1-2%; at the same time, parameter sets optimized for one dataset (columns named Best parameters) predictably vary a lot. By optimizing over all 6 datasets we have the same parameters set: \( d = 15, N = 500, k = 2 \), which is used for Europarl dataset in the final comparison methods.

| Dataset | 1-grams | 2-grams | 3-grams | 4-grams | Total candidates | Candidates among expected terms |
|---------|---------|---------|---------|---------|-----------------|---------------------------------|
| ACL 2.0| 763     | 520     | 65      | 6       | 1354            | 755                            |
| Patents| 1105    | 1105    | 290     | 47      | 2650            | 729                            |
| GENIA   | 5000    | 8536    | 2694    | 506     | 16736           | 9433                           |
| Krapivin| 36,665  | 153,625 | 44,488  | 7259    | 242,037         | 6038                           |
| FAO     | 44,835  | 201,685 | 52,004  | 7925    | 306,449         | 1343                           |
| ACL     | 91,026  | 236,001 | 65,195  | 7528    | 399,750         | 14,903                         |
| Europarl| 24,040  | 188,111 | 40,336  | 3292    | 255,779         | 6841                           |
To optimize parameters of PU-ATR method we perform grid search with the following set of possible values: coefficient used in ComboBasic method for the number of containing terms $a = \{0, 0.1, 0.5, 0.75, 1\}$; coefficient used in ComboBasic method for the number of contained terms $b = \{0, 0.1, 0.25, 0.5\}$; threshold used in PU algorithm for determining reliable negative instances $t = \{0.05, 0.025\}$.

Table 4 shows that parameters are not so stable, but the difference between test AvP and best AvP is about 1–2% in all cases. By optimizing over all 6 datasets we have the following parameter set: $a = 0.75$, $b = 0.5$, $t = 0.025$, which is used for Europarl dataset in the final comparison methods.

Tables 5 and 6 present comparison of all methods over all datasets. In order to estimate standard deviation, we use jackknife resampling: we split each dataset into 10 equal parts, or folds (16 for Patents, because it consists of 16 documents); then compute average precision on leave-one-out subsets of all folds. Note that we use parameter set chosen by cross-validation for KeyConceptRelatedness and PU-ATR and default parameters for other methods. Voting aggregates the same 5 features as PU-ATR, see Sect. 3.4.

PU-ATR seems to be the most stable: it is the best for 4 datasets and in top 3 methods for all datasets. However, it is the most computationally intensive method, see the next subsection. Basic and ComboBasic demonstrate good average

---

**Table 3** Best parameters for KeyConceptRelatedness method

| Test fold | Best parameters | Max AvP | CV parameters | Test AvP |
|-----------|-----------------|---------|---------------|---------|
| dNk       | d N k           |         |               |         |
| ACL 2.0   | 15 200 3        | 0.7645  | 15 500 2     | 0.7124  |
| Patents   | 15 50 3         | 0.6263  | 15 500 2     | 0.6190  |
| GENIA     | 3 500 3         | 0.6845  | 15 500 2     | 0.6757  |
| Krapivin  | 15 500 1        | 0.3623  | 15 300 2     | 0.2844  |
| FAO       | 20 300 2        | 0.4853  | 15 500 2     | 0.4671  |
| ACL       | 30 500 2        | 0.3294  | 15 500 2     | 0.3227  |

**Table 4** Best parameters for PU-ATR method

| Test fold | Best parameters | Max AvP | CV parameters | Test AvP |
|-----------|-----------------|---------|---------------|---------|
| abt       | a b t           |         |               |         |
| ACL 2.0   | 0.5 0.5 0.025   | 0.8137  | 1.0 0.1 0.1   | 0.8028  |
| Patents   | 0.75 0.5 0.025  | 0.6925  | 0.75 0.1 0.05 | 0.6821  |
| GENIA     | 0.5 0.5 0.05    | 0.7865  | 0.75 0.1 0.05 | 0.7823  |
| Krapivin  | 0.5 0.1 0.025   | 0.4389  | 0.75 0.1 0.1  | 0.4210  |
| FAO       | 0.75 0.1 0.1    | 0.4526  | 1.0 0.1 0.025 | 0.4429  |
| ACL       | 0.1 0.5 0.1     | 0.5089  | 0.75 0.1 0.05 | 0.4938  |
Table 5 All methods over 4 smallest datasets (average precision with standard deviation)

| Method          | ACL 2.0  | Patents  | GENIA       | Krapivin   |
|-----------------|----------|----------|-------------|------------|
| AvgTermFreq.    | 0.6787 ± 0.0029 | 0.5362 ± 0.0034 | 0.7081 ± 0.0006 | 0.1122 ± 0.0012 |
| ResidualIDF     | 0.6747 ± 0.0029 | 0.5227 ± 0.0032 | 0.7031 ± 0.0007 | 0.1070 ± 0.0013 |
| CValue          | 0.7836 ± 0.0009 | 0.6415 ± 0.0021 | 0.7272 ± 0.0005 | 0.3995 ± 0.0003 |
| Basic           | 0.6899 ± 0.0009 | 0.5508 ± 0.0036 | 0.6403 ± 0.0004 | 0.3878 ± 0.0006 |
| ComboBasic      | 0.6979 ± 0.0012 | 0.5486 ± 0.0036 | 0.6444 ± 0.0005 | 0.3887 ± 0.0007 |
| PostRankDC      | 0.6462 ± 0.0014 | 0.5022 ± 0.0028 | 0.6634 ± 0.0004 | 0.4042 ± 0.0018 |
| Relevance       | 0.7512 ± 0.0013 | 0.5030 ± 0.0054 | 0.7368 ± 0.0004 | 0.2992 ± 0.0003 |
| Weirdness       | 0.7552 ± 0.0014 | 0.5372 ± 0.0042 | 0.7651 ± 0.0004 | 0.3294 ± 0.0007 |
| NTM             | 0.7809 ± 0.0027 | 0.6104 ± 0.0029 | 0.7072 ± 0.0013 | 0.1053 ± 0.0020 |
| LinkProbability | 0.7101 ± 0.0021 | 0.4559 ± 0.0027 | 0.7045 ± 0.0007 | 0.1087 ± 0.0003 |
| KeyConceptRel.  | 0.7160 ± 0.0017 | 0.6131 ± 0.0023 | 0.6739 ± 0.0005 | 0.2893 ± 0.0004 |
| Voting          | 0.7868 ± 0.0012 | 0.6235 ± 0.0019 | 0.7546 ± 0.0005 | 0.2728 ± 0.0004 |
| PU-ATR          | **0.7938 ± 0.0022** | **0.6771 ± 0.0020** | **0.7760 ± 0.0013** | **0.4279 ± 0.0021** |

Bold indicates the best results taking into account the standard deviation.

Table 6 All methods over 3 biggest datasets (average precision with standard deviation)

| Method         | FAO       | ACL       | Europarl  |
|----------------|-----------|-----------|-----------|
| AvgTermFrequency | 0.0442 ± 0.0014 | 0.0698 ± 0.0004 | 0.1700 ± 0.0005 |
| ResidualIDF     | 0.0143 ± 0.0008 | 0.0660 ± 0.0004 | 0.1319 ± 0.0005 |
| CValue          | 0.3828 ± 0.0007 | 0.4292 ± 0.0002 | 0.3213 ± 0.0001 |
| Basic           | 0.3783 ± 0.0013 | 0.5362 ± 0.0003 | **0.3898 ± 0.0005** |
| ComboBasic      | 0.3787 ± 0.0013 | **0.5370 ± 0.0003** | **0.3894 ± 0.0005** |
| PostRankDC      | 0.4174 ± 0.0014 | 0.4492 ± 0.0012 | 0.3785 ± 0.0007 |
| Relevance       | 0.1513 ± 0.0011 | 0.4787 ± 0.0003 | 0.2134 ± 0.0003 |
| Weirdness       | 0.1450 ± 0.0017 | 0.4785 ± 0.0002 | 0.2283 ± 0.0003 |
| NovelTopicModel | 0.0588 ± 0.0038 | 0.2509 ± 0.0034 | 0.1999 ± 0.0030 |
| LinkProbability | 0.0075 ± 0.0002 | 0.1022 ± 0.0002 | 0.0897 ± 0.0002 |
| KeyConceptRelatedness | **0.4662 ± 0.0030** | 0.3274 ± 0.0005 | 0.3366 ± 0.0016 |
| Voting          | 0.1445 ± 0.0010 | 0.3401 ± 0.0010 | 0.2605 ± 0.0007 |
| PU-ATR          | 0.4397 ± 0.0021 | 0.4883 ± 0.0008 | 0.3681 ± 0.0010 |

Bold indicates the best results taking into account the standard deviation.

precision, particularly for large datasets. KeyConceptRelatedness is the best for FAO dataset only, most probably because Wikipedia has a good coverage of agriculture domain, which is especially relevant for small number of expected terms (1554 in FAO).

Note also that none of the methods showing best results in this experiment are implemented in other tools.
4.3 Performance results (time)

We estimated performance of ATR4S on a machine with Intel Core i5-2500 (3.3 GHz, 4 cores) and 32 Gb RAM, from which 12 Gb was set as a maximum memory allocation pool for Java Virtual Machine, see Table 7. Since preprocessing and candidates collection steps are the same for all methods, we show them in the first 2 rows and ignore that time for scoring/ranking methods.

As we can see, methods from the first 3 groups, i.e. those based on occurrences frequencies, contexts and reference corpora, are the fastest. Methods based on Wikipedia require constant 15 s (LinkProbability) or 1 min (KeyConceptRelatedness) for initialization, then their times depend on dataset size almost linearly. NovelTopicModel is the slowest for big datasets; however, its average precision is not good for big datasets anyway. Time required for PU-ATR is almost the sum of used features times and Spark start time (about 30 s).

5 Conclusion

This paper presents ATR4S, an open-source tool for automatic terms recognition, and experimental comparison of 13 state-of-the-art methods for ATR on 7 datasets. ATR4S comprises 13 state-of-the-art methods for ATR, supports caching and human-readable configuration; it is written in Scala with parallel collections wherever appropriate, so it utilizes all CPU cores.

Experimental comparison confirms the observation that no single method is best for all datasets (Zhang et al. 2008). However, we would suggest to try PU-ATR.
(Astrakhantsev 2015) for small datasets and/or the cases when time complexity is not an issue; Basic (Bordea et al. 2013) or ComboBasic (Astrakhantsev 2015)—for large datasets; KeyConceptRelatedness (Astrakhantsev 2015)—for tasks requiring few terms (compared to the dataset size) and supposing good coverage of the domain by Wikipedia. Also the survey and the experiments show that other available tools lack the best methods.

It is obvious that ATR4S does not include all methods capable of outperforming previously implemented methods on some settings, but we believe that these implementations can be used as a basis for development of other methods or, at least, for easy comparison. Nevertheless, the addition of new methods and their experimental evaluation are the main directions of the further improvement.

Regarding practical aspects of ATR task, in particular noisy input datasets, which often contain documents from multiple domains, and scenarios assuming terminology enrichment instead of extraction, we believe that incorporation of document clustering and more sophisticated semi-supervised methods are among the most promising research topics.

Acknowledgements The author would like to thank Yaroslav Nedumov and Denis Turdakov for their valuable comments and Denis Fedorenko for his help in implementing the previous versions of ATR tool.

References

Ahmad, K., Gillam, L., Tostevin, L., et al. (1999). University of Surrey participation in TREC8: Weirdness indexing for logical document extrapolation and retrieval (WILDER). In Proceedings of the eighth text retrieval conference (TREC-8).

Astrakhantsev, N. (2014). Automatic term acquisition from domain-specific text collection by using Wikipedia. Proceedings of the institute for system programming, 26(4), 7–20.

Astrakhantsev, N. (2015). Methods and software for terminology extraction from domain-specific text collection. Ph.D. thesis, Institute for System Programming of Russian Academy of Sciences.

Astrakhantsev, N., Fedorenko, D., & Turdakov, D. (2014). Automatic enrichment of informal ontology by analyzing a domain-specific text collection. In Computational linguistics and intellectual technologies: Papers from the annual international conference dialogue (Vol. 13, pp. 29–42).

Astrakhantsev, N., Fedorenko, D., & Turdakov, D. Y. (2015). Methods for automatic term recognition in domain-specific text collections: A survey. Programming and Computer Software, 41(6), 336–349.

Bordea, G., Buitelaar, P., & Polajnar, T. (2013). Domain-independent term extraction through domain modelling. In The 10th international conference on terminology and artificial intelligence (TIA 2013), Paris, France.

Buitelaar, P., & Eigner, T. (2009). Expertise mining from scientific literature. In Proceedings of the fifth international conference on knowledge capture, K-CAP ’09 (pp. 171–172). New York, NY: ACM.

Church, K., & Gale, W. (1999). Inverse document frequency (IDF): A measure of deviations from poisson. In Natural language processing using very large corpora (pp. 283–295). Springer.

Cram, D., & Daille, B. (2016). Termsuite: Terminology extraction with term variant detection. In Proceedings of ACL 2016 (p. 13).

El-Beltagy, S.R., & Rafea, A. (2010). Kp-miner: Participation in SEMEVAL-2. In Proceedings of the 5th international workshop on semantic evaluation (pp. 190–193). Association for Computational Linguistics.

Evans, D. A., & Lefferts, R. G. (1995). Clarit-TREC experiments. Information Processing & Management, 31(3), 385–395.

Faralli, S., & Navigli, R. (2013). Growing multi-domain glossaries from a few seeds using probabilistic topic models. In EMNLP (pp. 170–181)
Fedorenko, D., Astrakhantsev, N., & Turdakov, D. (2013). Automatic recognition of domain-specific terms: An experimental evaluation. In *Proceedings of SYRCoDIS 2013* (pp. 15–23).

Frantzi, K., Ananiadou, S., & Mima, H. (2000). Automatic recognition of multi-word terms: The C-value/NC-value method. *International Journal on Digital Libraries, 3*(2), 115–130.

Gaussier, É. (1998) Flow network models for word alignment and terminology extraction from bilingual corpora. In *Proceedings of the 17th international conference on computational linguistics* (Vol. 1, pp. 444–450). Association for Computational Linguistics.

Harris, Z. S. (1954). Distributional structure. *Word, 10*(2–3), 146–162.

Judea, A., Schütze, H., & Bruegmann, S. (2014). Unsupervised training set generation for automatic acquisition of technical terminology in patents. In *Proceedings of COLING 2014, the 25th international conference on computational linguistics: Technical Papers* (pp. 290–300). Dublin City University and Association for Computational Linguistics, Dublin, Ireland.

Kageura, K., & Umino, B. (1996). Methods of automatic term recognition: A review. *Terminology, 3*(2), 259–289.

Kim, J. D., Ohta, T., Tateisi, Y., & Tsujii, J. (2003). GENIA corpus—a semantically annotated corpus for bio-textmining. *Bioinformatics, 19*(Suppl 1), i180–i182.

Koehn, P. (2005). Europarl: A parallel corpus for statistical machine translation. In *MT summit* (Vol. 5, pp. 79–86). http://mt-archive.info/MTS-2005-Koehn.pdf.

Kozakov, L., Park, Y., Fin, T., Drissi, Y., Doganata, Y., & Cofino, T. (2004). Glossary extraction and utilization in the information search and delivery system for ibm technical support. *IBM Systems Journal, 43*(3), 546–563.

Krapivin, M., Autaeu, A., & Marchese, M. (2009). Large dataset for keyphrases extraction. http://eprints.biblio.unitn.it/1671/1/disio09055-krapivin-autayeu-marchese.pdf.

Li, S., Li, J., Song, T., Li, W., & Chang, B. (2013). A novel topic model for automatic term extraction. In *Proceedings of the 36th international ACM SIGIR conference on research and development in information retrieval* (pp. 885–888). ACM.

Lingpeng, Y., Donghong, J., Guodong, Z., & Yu, N. (2005). Improving retrieval effectiveness by using key terms in top retrieved documents. In D. E. Losada & J. M. Fernandez-Luna (eds) *Advances in information retrieval ECIR 2005*. Lecture Notes in Computer Science (Vol. 3408). Berlin: Springer. https://link.springer.com/chapter/10.1007/978-3-540-31865-1_13#citeas.

Liu, B., Lee, W. S., Yu, P. S., & Li, X. (2002). Partially supervised classification of text documents. In *Proceedings of ICML* (Vol. 2, pp. 387–394).

Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S. J., & McClosky, D. (2014). The Stanford CoreNLP natural language processing toolkit. In *Proceedings of the 52nd Annualmeeting of the association for computational linguistics: System Demonstrations* (pp. 55–60). https://stanfordnlp.github.io/CoreNLP/.

Mayorov, V., Andrianov, I., Astrakhantsev, N., Avanesov, V., Kozlov, I., & Turdakov, D. (2015). A high precision method for aspect extraction in russian. In *Proceedings of international conference extraction in russian* (Vol. 2).

Medelyan, O., & Witten, I. H. (2008). Domain-independent automatic keyphrase indexing with small training sets. *Journal of the American Society for Information Science and Technology, 59*(7), 1026–1040.

Meijer, K., Frasinacar, F., & Hogenboom, F. (2014). A semantic approach for extracting domain taxonomies from text. *Decision Support Systems, 62*, 78–93.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, & K. Q. Weinberger (Eds.) *Proceedings of the 26th International Conference on Neural Information Processing Systems* (Vol. 2, pp. 3111–3119). https://dl.acm.org/citation.cfm?id=2999959.

Mondary, T., Nazarenko, A., Zargayouna, H., & Barreaux, S. (2012) The quaero evaluation campaign on term extraction. In *Proceedings of the eighth international conference on language resources and evaluation (LREC)* (pp. 663–669).

Nokel, M., & Loukachevitch, N. (2013). An experimental study of term extraction for real information-retrieval thesauri. In *Proceedings of 10th international conference on terminology and artificial intelligence* (pp. 69–76).

Park, Y., Byrd, R., & Boguraev, B. (2002). Automatic glossary extraction: Beyond terminology identification. In *Proceedings of the 19th international conference on computational linguistics* (Vol. 1, pp. 1–7). Association for Computational Linguistics.
Pazienza, M. T., Pennacchiotti, M., & Zanzotto, F. M. (2005). Terminology extraction: An analysis of linguistic and statistical approaches. In S. Sirmakessis (Eds.) Knowledge mining Studies in Fuzziness and Soft Computing (Vol. 185). Berlin, Heidelberg: Springer. https://link.springer.com/chapter/10.1007/3-540-32394-5_20#citeas.

Peñas, A., Verdejo, F., & Gonzalo, J., et al. (2001). Corpus-based terminology extraction applied to information access. In Proceedings of corpus linguistics (Vol. 2001).

Qasemizadeh, B., & Schumann, A.K. (2016). The ACL RD-TEC 2.0. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics, Charles University in Prague. http://hdl.handle.net/11372/LRT-1661.

Salton, G. (1971). The SMART retrieval system—experiments in automatic document processing. Upper Saddle River: Prentice-Hall Inc.

Spasić, I., Greenwood, M., Preece, A., Francis, N., & Elwyn, G. (2013). Flexiterm: A flexible term recognition method. Journal of Biomedical Semantics, 4(1), 1.

Turdakov, D. Y., Astrakhantsev, N., Nedumov, Y. R., Sysoev, A., Andrianov, I., Mayorov, V., et al. (2014). Texterra: A framework for text analysis. Programming and Computer Software, 40(5), 288–295.

Ventura, J. A. L., Jonquet, C., Roche, M., & Teisseire, M., et al. (2013). Combining C-value and keyword extraction methods for biomedical terms extraction. In LBM’2013: International symposium on languages in biology and medicine (pp. 45–49).

Vivaldi, J., Cabrera-Diego, L. A., Sierra, G., & Pozzi, M. (2012). Using Wikipedia to validate the terminology found in a corpus of basic textbooks. In LREC (pp. 3820–3827).

Vivaldi, J., & Rodríguez, H. (2010). Using Wikipedia for term extraction in the biomedical domain: first experiences. Procesamiento del Lenguaje Natural, 45, 251–254.

Vivaldi, J., & Rodríguez, H. (2011). Extracting terminology from Wikipedia. Procesamiento del lenguaje natural, 47, 65–73.

Wermter, J., & Hahn, U. (2006). You can’t beat frequency (unless you use linguistic knowledge): a qualitative evaluation of association measures for collocation and term extraction. In Proceedings of the 21st international conference on computational linguistics and the 44th annual meeting of the association for computational linguistics (pp. 785–792). Association for Computational Linguistics.

Wu, W., Liu, T., Hu, H., Du, X. (2012). Extracting domain-relevant term using Wikipedia based on random walk model. In ChinaGrid annual conference (ChinaGrid), 2012 seventh (pp. 68–75). IEEE.

Zadeh, B. Q., 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’14), Dublin, Ireland.

Zhang, Z., Brewster, C., & Ciravegna, F. (2008). A comparative evaluation of term recognition algorithms. In Proceedings of the sixth international conference on language resources and evaluation (LREC08), Marrakech, Morocco.

Zhang, Z., Gao, J., & Ciravegna, F. (2016). JATE 2.0: Java automatic term extraction with Apache Solr. In The LREC 2016 proceedings.