CHAPTER NUMBER

Corpus-Based Terminology Extraction

ALEXANDRE PATRY AND PHILIPPE LANGLAIS

Terminology management is a key component of many natural language processing activities such as machine translation (Langlais and Carl, 2004), text summarization and text indexation. With the rapid development of science and technology continuously increasing the number of technical terms, terminology management is certain to become of the utmost importance in more and more content-based applications.

While the automatic identification of terms from texts has been the focus of past studies (Jacquemin, 2001) (Castellví et al, 2001), the current trend in Terminology Management (TM) has shifted to the issue of term networking (Kageura et al, 2004). A possible explanation of this shifting may lie in the fact that Terminology Extraction (TE), although being a noisy activity, encompasses well established techniques that seem difficult to improve significantly upon.

Despite this shift, we do believe that better extraction of terms could carry over subsequent steps of TM. A traditional TE system usually involves a subtle mixture of linguistic rules and statistical metrics in order to identify a list of candidate terms where it is hoped that terms are ranked first.

We distinguish our approach to TE from traditional ones in two different ways. First, we give back to the user an active role in the extraction process. That is, instead of encoding a static definition of what might or might not be a term, we let the user specify his own. We do so by asking him to set up a training corpus (a corpus where the terms have been identified by a human) from which our extractor will learn how to define a term. Second, our approach is completely automatic and is readily adapted to the tools (part-of-speech tagger,

lemmatizer) and metrics of the user.

One might object that requiring a training corpus is asking the user to do a part of the job the machine is supposed to do, but we see it in a different way. We consider that a little help from the user could pay back in flexibility.

The structure of our paper outlines the three steps involved in our approach. In the following section, we describe our algorithm to identify candidate terms. In the third section, we introduce the different metrics we compute to score them. The fourth section explains how we applied AdaBoost (Freund and Schapire, 1999), a machine learning algorithm, to rank and identify a list of terms. We then evaluate our approach on a corpus which was set up by the *Office québécois de la langue française* to evaluate commercially available term extractors. We show that our classifier outperforms the individual metrics used in this study. Finally, we discuss some limitations of our approach and propose future works to be done.

Extraction of candidate terms

It is a common practice to extract candidate terms using a part-of-speech (POS) tagger and an automaton (a program extracting word sequences corresponding to predefined POS patterns). Usually, those patterns are manually handcrafted and target noun phrases, since most of the terms of interest are noun phrases (Justeson and Katz, 1995). Typical examples of such patterns can be found in (Jacquemin, 2001).

As pointed out in (Justeson and Katz, 1995), relying on a POS tagger and legitimate pattern recognition is error prone, since taggers are not perfect. This might be especially true for very domain specific texts where a tagger is likely to be more erratic. To overcome this problem without giving up the use of POS patterns (since they are easy to design and to use), we propose a way to use a training corpus in order to automate the creation of an automaton.

There are many potential advantages with this approach. First, the POS tagger and the tagging errors, to the extent that they are consistent, will be automatically assimilated by the automaton. Second, this gives to the user the opportunity to specify the terms that are of interest for him. If many terms involving verbs are found in the training corpus, the automaton will reflect that interest as well. We also observed in informal experiments that wide spread patterns often fails to extract many terms found in our training corpus.

Several approaches can be applied when generating an automaton from sequences of POS encountered in a training corpus. A

straightforward approach is to memorize all the sequences seen in the training corpus. A sequence of words is thus a candidate term only if its sequence of POS tags has been seen before. This approach is simple but naive. It cannot generate new patterns that are slight variations of the ones seen at training time, and an isolated tagging error can lead to a bad pattern.

To avoid those problems, we propose to generate the patterns using a language model trained on the POS tags of the terms found in the training corpus. A language model is a function computing the probability that a sequence of words has been generated by a certain language. In our case, the words are POS tags and the language is the one recognizing the sequences of tags corresponding to terms. Our language model can be described as follow:

$$P(w_1^n) = \prod_{i=1}^{n} P(w_i \mid H_i)$$

where w_1^n is a sequence of POS tags and H_i is called the history which summarizes the information of the $i-1$ previous tags. To build an automaton, we only have to set a threshold and generate all the patterns whose probability is higher than it. An excerpt of such an automaton is given in Figure 1.

Probability	Pattern
0.538	NomC AdjQ
0.293	NomC Prep NomC
0.032	NomC Dete-dart-ddef NomC
0.0311	NomC Verb-ParPas
0.0311	NomC Prep Dete-dart-ddef NomC
…	

Figure 1 Excerpt of an automatically generated automaton.

Another advantage of such an automaton is that all patterns are associated with a probability, giving more information than a binary value (legitimate or not). Indeed, the POS pattern probability is one of the numerous metrics that we feed our classifier with.

Scoring the candidate terms

In the previous section, we showed a way to generate an automaton that extracts a set of candidate terms that we now want to rank and/or filter. Following many other works on term extraction, we score each

candidate using various metrics. Many different ones have been identified in (Daille, 1994) and (Castellví et al, 2001). We do not believe that a single metric is sufficient, but instead think that it is more fruitful to use several of them and train a classifier to learn how to take benefit of each of them.

Because we think they are interesting for the task, we retained the following metrics: the frequency, the length, the log-likelyhood, the entropy, tf·idf and the POS pattern probabilities discussed in the previous section. Recall however that our approach is not restricted to these metrics, but instead can benefit from any other one that can be computed automatically.

Alone, the frequency is not a robust metric to assess the terminological property of a candidate, but it does carry useful information, as does also the length of terms.

In (Dunning, 1993), Dunning advocates the use of log-likelyhood to measure whether two events that occur together do so as a coincidence or not. In our case, we want to measure the cohesion of a complex candidate term (a candidate term composed of two words or more) by verifying if its words occur together as a coincidence or not. The log-likelyhood ratio of two adjacent words (u and v) can be computed with the following formula (Daille, 1994):

$$\begin{aligned} -\log uv = \; & a\log a + b\log b + c\log c + d\log d + N\log N \\ & -(a+c)\log(a+c) - (a+b)\log(a+b) \\ & -(c+d)\log(c+d) - (d+b)\log(d+b) \end{aligned}$$

where a is the number of times uv appears in the document, b the number of times u appears not followed by v, c the number of times v appears not preceded by u, N the corpus size and d the number of candidate terms that does not involve u or v. Following (Russell, 1998), to compute log-likelyhood on candidate terms involving more than two words, we keep the minimum value among the log-likelyhood of each possible split in the candidate term.

With the intuition that terms are coherent units that can appear surrounded by various different words, we use as well the entropy to rate a candidate term. The entropy of a candidate is computed by averaging its left and right entropy:

$$e(w_1^n) = \frac{e_{left}(w_1^n) + e_{right}(w_1^n)}{2}$$

$$e_{left}(s) = \sum_{\{u:us\in C\}} h\left(\frac{|us|}{|s|}\right)$$

$$e_{right}(s) = \sum_{\{u:su \in C\}} h\left(\frac{|su|}{|s|}\right)$$

$$h(x) = x \log x$$

where w_1^n is the candidate term and C is the corpus from which we are extracting the terms.

Finally, to weight the salience of a candidate term, we also use tf·idf. This metric is based on the idea that terms describing a document should appear often in it but should not appear in many other documents. It is computed by dividing the frequency of a candidate term by the number of documents in an out-of-domain corpus that contains it. Because tf·idf is usually computed on one word, when we evaluated complex candidate terms, we computed tf·idf on each of its words and kept five values: the first, the last, the minimum, the maximum and the average. In our experiments, the out-of-domain corpus was composed of texts taken from the French Canadian parliamentary debates (the so-called Hansard), totalizing 1.4 million sentences.

Identifying terms among candidates

Once each candidate terms is scored, we must decide which ones should finally be elected a term. To accomplish this task, we train a binary classifier (a function which qualifies a candidate as a term or not) on the face of the scores we computed for a candidate.

We use the AdaBoost learning algorithm (Freund and Schapire, 1999) to build this classifier. AdaBoost is a simple but efficient learning technique that combines many weak classifiers (a weak classifier must be right more than half of the time) into a stronger one. To achieve this, it trains them successively, each time focusing on examples that have been hard to classify correctly by the previous weak classifiers. In our experiments, the weak classifiers were binary stumps (binary classifiers that compare one of the score to a given threshold to classify a candidate term) and we limited their number to 50. An example of such a classifier is presented in Figure 2.

Experiments

Our community lacks a common benchmark on which we could compare our result with others. In this work, we applied our approach to a corpus called EAU. It is composed of six texts dealing with water supply. Its complex terms have been listed by some members or the

Office québécois de la langue française for a project called ATTRAIT (*Atelier de Travail Informatisé du Terminologue*) whose main objective was to evaluate existing software solutions for the terminologist[1].

Input: A scored candidate term *c*

$\beta = 0$
if entropy(*c*) > 1.6 then $\beta = \beta + 0.26$ else $\beta = \beta - 0.26$
if length(*c*) > 1.6 then $\beta = \beta + 0.08$ else $\beta = \beta - 0.08$
…
if $\beta > 0$ then return *term* else return *not-term*

Figure 2 An excert from a classifier generated by the Adaboost learning algorithm.

In our experiments, we kept the preprocessing stage as simple as possible. The corpus and the list of terms were automatically tokenized, lemmatized and had their POS tagged with an in-house package (Foster, 1991). Once preprocessed, the EAU corpus is composed 12 492 words and 208 terms. Of these 208 terms, 186 appear without syntactic variation (as they were listed) a total of 400 times.

Since the terms of our evaluation corpus are already identified, it is straightforward to compute the precision and the recall of our system. Precision (resp. recall) is the ratio of terms correctly identified by the system over the total number of terms identified as such (resp. over the total number of terms manually identified in the list).

We evaluated our system using five fold cross-validation. This means that the corpus was partitioned into five subsets and that five experiments were run each time testing with a different subset and training the automaton and the classifier with the four others. Each training set (resp. testing set) was composed of about 12 000 (resp. 3000) words containing an average of about 150 (resp. 50) terms.

Because only complex terms are listed and because we do not consider term variations, our results only consider complex terms that appear without variation. Also, after informal experiments, we set the minimum probability of a pattern to be accepted by our automaton to 0.005. The performance of our system, averaged on the five fold of the cross-validation, can be found in Table 1.

From the results, we can see that the automaton has a high recall but a low precision, which was to be expected. Indeed, the automaton is only

1. See http://www.rint.org for more details on this project.

a rough filter that eliminates easy to eliminate word sequences, but keep as much terms as possible. On the other hand, the selection did not perform as well as we expected. Its low recall and precision could be explained by the metrics that are not as expressive as we though and by the fact that 75% of the terms in our test corpora appears only one time. When a term appears only one time, its frequency and entropy become useless. The results presented in Table 2 seem to confirm our hypothesis.

	Part	μ	σ
Extraction	Precision	0.14	0.05
	Recall	0.94	0.03
Identification	Precision	0.45	0.19
	Recall	0.41	0.20
Overall system	Precision	0.43	0.18
	Recall	0.38	0.18

Table 1 Mean (μ) and standard deviation (σ) of the precision and recall of the different parts of our system.

Because we wanted to compare our system with the individual metrics that it uses, we had to modify it such that it ranks the candidate terms instead of simply accepting or rejecting them. To do so, we made our system return β instead of *term* or *not term* (see Figure 2). We then sorted the candidate terms in decreasing order of their β value.

A common practice when comparing ranking algorithms is to build their ROC (receiving operator curve), which shows the ratio of good identifications (y axis) against the ratio of bad identification (x axis) for all the acceptation thresholds. The best curve will augment in y faster than in x, so will have a greater area under it. We can see in Figure 3 that our system performs better than entropy or log-likelyhood alone. This leads us to believe that different scores carry different information and that combining them, as we did it, is fruitful.

Discussion and future works

In this paper, we presented an approach to automatically generate an end-to-end term extractor from a training corpus. We also proposed a way to combine many statistical scores in order to extract terms more efficiently than when each score is used in isolation.

Because of the nature of the training algorithm, we can easily extend

the set of metrics we considered here. Even a priori knowledge could be integrated by specifying keywords before the extraction and setting a score to one when a candidate term contains a keyword or zero otherwise. The same flexibility is achieved when the automaton is created. By generating it directly from the output of the POS tagger, our solution does not depend of a particular tagger and is tolerant to consistent tagging errors.

Criteria		μ	σ
Candidates appearing one time	Precision	0.39	0.16
	Recall	0.33	0.22
Candidates appearing at least two times	Precision	0.73	0.14
	Recall	0.85	0.09

Table 2 Comparison of the performance of the term identification part for candidates appearing with different frequencies.

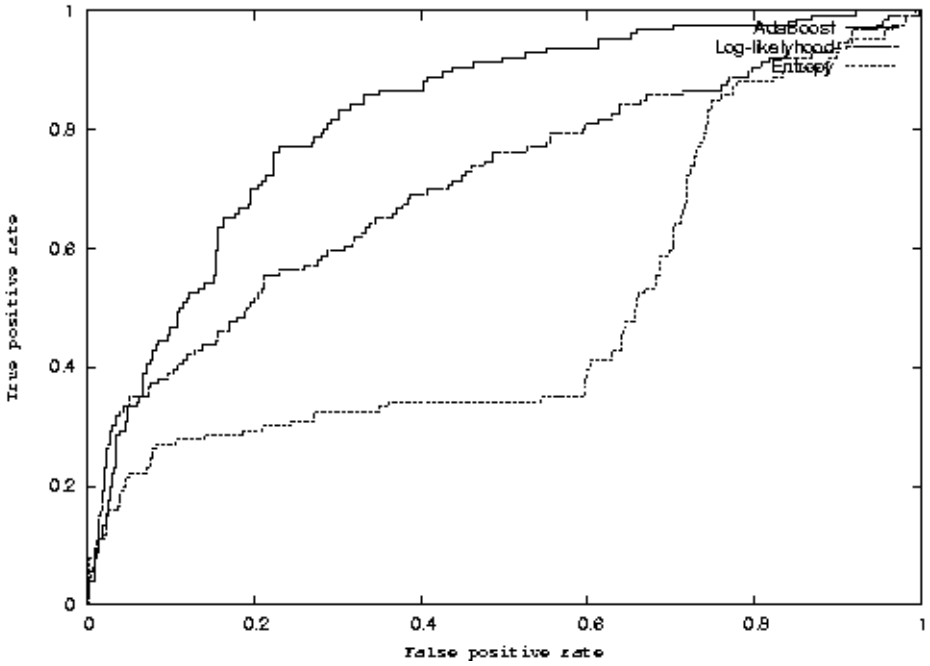

Figure 3 The ROC of our system (AdaBoost) against two other score when we trained our system on one half of our corpus and tested on the other. A greater area under the curve is better.

A shortcoming of this work is that we did not treat term variations. Terminology variation is a well-known phenomenon, whose amount is estimated according to (Kageura et al., 2004) from 15% to 35%. We think that the best way to deal with them in our framework would be to

introduce a preprocessing stage where variations are normalized to a canonical form. Term variations have been extensively studied in (Jacquemin, 2001) and (Daille, 2003).

In our experiments, we focused on complex terms. Because some scores do not apply to simple terms (*e.g.* log-likelyhood and length), we think that the best way to extract simple terms would be to train a dedicated classifier.

Acknowledgements

We would like to thank Hugo Larochelle who found the corpus we used in our experiments and Elliott Macklovitch who made some useful comments on the first draft of this document. This work has been subsidized by NSERC and FQRNT.

References

Castellví, M. Teresa Cabré; Bagot, Rosa Estopà; Palastresi, Jordi Vivaldi; Automatic Term Detection: A Review of Current Systems in *Recent advances in computational terminology*. John Benjamin, 2001.

Daille, Béatrice; Study and Implementation of Combined Techniques for Automatic Extraction of Terminology in *The Balancing Act: Combining Symbolic and Statistical Approaches to Language*. New Mexico State University, Las Cruces, 1994.

Daille, Béatrice; Conceptual structuring through term variations in *Proceedings of the ACL Workshop on Multiword Expressions: Analysis, Acquisition and Treatment*. 2003.

Dunning, Ted; *Accurate Methods for the Statistics of Surprise and Coincidence*. 1993.

Foster, George; *Statistical lexical disambiguation*, Master Thesis. McGill University, Montreal, 1991.

Freund, Y.; Schapire, R.E.; A Short Introduction to Boosting in *Journal of Japanese Society for Artificial Intelligence*. 1999.

Jacquemin, Christian; *Spotting and Discovering Terms through Natural Language Processing*. MIT Press, 2001.

Justeson, John S.; Katz, Slava M.; Technical Terminology: Some Linguistic Properties and an Algorithm for Identification in Text in *Natural Language Engineering*. 1995.

Kageura, Kyo; Daille, Béatrice; Nakagawa, Hiroshi; Chien, Lee-Feng; Recent Trends in Computational Terminology in *Terminology*. John Benjamin, 2004.

Langlais, Philippe; Carl, Michael; General-purpose statistical translation engine and domain specific texts: Would it work? in *Terminology*. John Benjamin, 2004.