Minimally Supervised Learning of Semantic Knowledge from Query Logs

Mamoru Komachi
Nara Institute of Science and Technology
8916-5 Takayama
Ikoma, Nara 630-0192, Japan
mamoru-k@is.naist.jp

Hisami Suzuki
Microsoft Research
One Microsoft Way
Redmond, WA 98052 USA
hisamis@microsoft.com

Abstract

We propose a method for learning semantic categories of words with minimal supervision from web search query logs. Our method is based on the *Espresso* algorithm (Pantel and Pennacchiotti, 2006) for extracting binary lexical relations, but makes important modifications to handle query log data for the task of acquiring semantic categories. We present experimental results comparing our method with two state-of-the-art minimally supervised lexical knowledge extraction systems using Japanese query log data, and show that our method achieves higher precision than the previously proposed methods. We also show that the proposed method offers an additional advantage for knowledge acquisition in an Asian language for which word segmentation is an issue, as the method utilizes no prior knowledge of word segmentation, and is able to harvest new terms with correct word segmentation.

1 Introduction

Extraction of lexical knowledge from a large collection of text data with minimal supervision has become an active area of research in recent years. Automatic extraction of relations by exploiting recurring patterns in text was pioneered by Hearst (1992), who describes a bootstrapping procedure for extracting words in the hyponym (*is-a*) relation, starting with three manually given lexico-syntactic patterns. This idea of learning with a minimally supervised bootstrapping method using surface text patterns was subsequently adopted for many tasks, including relation extraction (e.g., Brin, 1998; Riloff and Jones, 1999; Pantel and Pennacchiotti, 2006) and named entity recognition (e.g., Collins and Singer, 1999; Etzioni et al., 2005).

In this paper, we describe a method of learning semantic categories of words using a large collection of Japanese search query logs. Our method is based on the *Espresso* algorithm (Pantel and Pennacchiotti, 2006) for extracting binary lexical relations, adapting it to work well on learning unary relations from query logs. The use of query data as a source of knowledge extraction offers some unique advantages over using regular text.

- Web search queries capture the interest of search users directly, while the distribution of the Web documents do not necessarily reflect the distribution of what people search (Silverstein et al., 1998). The word categories acquired from query logs are thus expected to be more useful for the tasks related to search.
- Though user-generated queries are often very short, the words that appear in queries are generally highly relevant for the purpose of word classification.
- Many search queries consist of *keywords*, which means that the queries include word segmentation specified by users. This is a great source of knowledge for learning word boundaries for those languages whose regularly written text does not indicate word boundaries, such as Chinese and Japanese.

Although our work naturally fits into the larger goal of building knowledge bases automatically from text, to our knowledge we are the first to explore the use of Japanese query logs for the purpose of minimally supervised semantic category acquisition. Our work is similar to Sekine and Suzuki (2007), whose goal is to augment a manually created dictionary of named entities by finding

	# of seed	Target	# of iteration	Corpus	Language
Sekine & Suzuki	~600	Categorized NEs	1	Query log	English
Basilisk	10	Semantic lexicon	∞	MUC-4	English
Espresso	~10	Semantic relations	∞	TREC	English
Tchai	5	Categorized words	∞	Query log	Japanese

Table 1: Summary of algorithms

contextual patterns from English query logs. Our work is different in that it does not require a full-scale list of categorized named entities but a small number of seed words, and iterates over the data to extract more patterns and instances. Recent work by Paşca (2007) and Paşca and Van Durme (2007) also uses English query logs to extract lexical knowledge, but their focus is on learning attributes for named entities, a different focus from ours.

2 Related Work

In this section, we describe three state-of-the-art algorithms of relation extraction, which serve as the baseline for our work. They are briefly summarized in Table 1. The goal of these algorithms is to learn target *instances*, which are the words belonging to certain categories (e.g., *cat* for the Animal class), or in the case of relation extraction, the pairs of words standing in a particular relationship (e.g., *pasta*::*food* for *is-a* relationship), given the *context patterns* for the categories or relation types found in source data.

2.1 Pattern Induction

The first step toward the acquisition of instances is to extract context patterns. In previous work, these are surface text patterns, e.g., *X such as Y*, for extracting words in an *is-a* relation, with some heuristics for finding the pattern boundaries in text. As we use query logs as the source of knowledge, we simply used everything but the instance string in a query as the pattern for the instance, in a manner similar to Paşca et al. (2006). For example, the seed word *JAL* in the query "*JAL+flight_schedule*" yields the pattern "*#+flight_schedule*".[1] Note that we perform no word segmentation or boundary detection heuristics in identifying these patterns, which makes our approach fast and robust, as the segmentation errors introduce noise in extracted patterns, especially when the source data contains many out of vocabulary items.

The extracted context patterns must then be assigned a score reflecting their usefulness in extracting the instances of a desired type. Frequency is a poor metric here, because frequent patterns may be extremely *generic*, appearing across multiple categories. Previously proposed methods differ in how to assign the desirability scores to the patterns they find and in using the score to extract instances, as well as in the treatment of generic patterns, whose precision is low but whose recall is high.

2.2 Sekine and Suzuki (2007)'s Algorithm

For the purpose of choosing the set of context patterns that best characterizes the categories, Sekine and Suzuki (2007) report that none of the conventional co-occurrence metrics such as tf.idf, mutual information and chi-squared tests achieved good results on their task, and propose a new measure, which is based on the number of different instances of the category a context *c* co-occurs with, lized by its token frequency for all categories:

$$Score(c) = f_{type} \log \frac{g(c)}{C}$$

$$g(c) = f_{type}(c) / F_{inst}(c)$$

$$C = f_{type}(ctop1000) / F_{inst}(ctop1000)$$

where f_{type} is the type frequency of instance terms that *c* co-occurs with in the category, F_{inst} is the token frequency of context *c* in the entire data and *ctop1000* is the 1000 most frequent contexts. Since they start with a large and reliable named entity dictionary, and can therefore use several hundred seed terms, they simply used the top-*k* highest-scoring contexts and extracted new named entities once and for all, without iteration. Generic patterns receive low scores, and are therefore ignored by this algorithm.

2.3 The *Basilisk* Algorithm

Thelen and Riloff (2002) present a framework called *Basilisk*, which extracts semantic lexicons

[1] # indicates where the instance occurs in the query string, and + indicates a white space in the original Japanese query. The underscore symbol (_) means there was originally no white space; it is used merely to make the translation in English more readable.

[2] The manual classification assigns only one category

for multiple categories. It starts with a small set of seed words and finds all patterns that match these seed words in the corpus. The bootstrapping process begins by selecting a subset of the patterns by the *R*log*F* metric (Riloff, 1996):

$$R\log F(pattern_i) = \frac{F_i}{N_i} \cdot \log(F_i)$$

where F_i is the number of category members extracted by $pattern_i$ and N_i is the total number of instances extracted by $pattern_i$. It then identifies instances by these patterns and scores each instance by the following formula:

$$AvgLog(word_i) = \frac{\sum_{j=1}^{P_i} \log(F_j + 1)}{P_i}$$

where P_i is the number of patterns that extract $word_i$. They use the average logarithm to select instances to balance the recall and precision of generic patterns. They add five best instances to the lexicon according to this formula, and the bootstrapping process starts again. Instances are cumulatively collected across iterations, while patterns are discarded at the end of each iteration.

2.4 The *Espresso* Algorithm

We will discuss the *Espresso* framework (Pantel and Pennacchiotti, 2006) in some detail because our method is based on it. It is a general-purpose, minimally supervised bootstrapping algorithm that takes as input a few seed instances and iteratively learns surface patterns to extract more instances. The key to *Espresso* lies in its use of generic patterns: Pantel and Pennacchiotti (2006) assume that correct instances captured by a generic pattern will also be instantiated by some *reliable patterns*, which denote high precision and low recall patterns.

Espresso starts from a small set of seed instances of a binary relation, finds a set of surface patterns *P*, selects the top-*k* patterns, extracts the highest scoring *m* instances, and repeats the process. *Espresso* ranks all patterns in *P* according to reliability r_π, and retains the top-*k* patterns for instance extraction. The value of *k* is incremented by one after each iteration.

The reliability of a pattern *p* is based on the intuition that a reliable pattern co-occurs with many reliable instances. They use pointwise mutual information (PMI) and define the reliability of a pattern *p* as its average strength of association across each input instance *i* in the set of instances *I*, weighted by the reliability of each instance *i*:

$$r_\pi(p) = \frac{\sum_{i \in I} \left(\frac{pmi(i,p)}{\max_{pmi}} \cdot r_i(i) \right)}{|I|}$$

where $r_i(i)$ is the reliability of the instance *i* and $\max_{pmi}$ is the maximum PMI between all patterns and all instances. The PMI between instance $i = \{x,y\}$ and pattern *p* is estimated by:

$$pmi(i,p) = \log \frac{|x,p,y|}{|x,*,y||*,p,*|}$$

where $|x,p,y|$ is the frequency of pattern *p* instantiated with terms *x* and *y* (recall that *Espresso* is targeted at extracting binary relations) and where the asterisk represents a wildcard. They multiplied $pmi(i,p)$ with the discounting factor suggested in Pantel and Ravichandran (2004) to alleviate a bias towards infrequent events.

The reliability of an instance is defined similarly: a reliable instance is one that associates with as many reliable patterns as possible.

$$r_i(i) = \frac{\sum_{p \in P} \left(\frac{pmi(i,p)}{\max_{pmi}} \cdot r_\pi(p) \right)}{|P|}$$

where $r_\pi(p)$ is the reliability of pattern *p*, and *P* is the set of surface patterns. Note that $r_i(i)$ and $r_\pi(p)$ are recursively defined: the computation of the pattern and instance reliability alternates between performing pattern reranking and instance extraction. Similarly to *Basilisk*, instances are cumulatively learned, but patterns are discarded at the end of each iteration.

3 The *Tchai* Algorithm

In this section, we describe the modifications we made to *Espresso* to derive our algorithm called *Tchai*.

3.1 Filtering Ambiguous Instances and Patterns

As mentioned above, the treatment of high-recall, low-precision generic patterns (e.g., *#+map*, *#+animation*) present a challenge to minimally supervised learning algorithms due to their amguity. In the case of semantic category acquisition, the problem of ambiguity is exacerbated, because not only the acquired patterns, but also the instances can be highly ambiguous. For example,

once we learn an ambiguous instance such as *Pokemon*, it will start collecting patterns for multiple categories (e.g., Game, Animation and Movie), which is not desirable.

In order to control the negative effect of the generic patterns, *Espresso* introduces a confidence metric, which is similar but separate from the reliability measure, and uses it to filter out the generic patterns falling below a confidence threshold. In our experiments, however, this metric did not produce a score that was substantially different from the reliability score. Therefore, we did not use a confidence metric, and instead opted for not ing ambiguous instances and patterns, where we define *ambiguous instance* as one that induces more than 1.5 times the number of patterns of viously accepted reliable instances, and *ambiguous* (or *generic*) *pattern* as one that extracts more than twice the number of instances of previously accepted reliable patterns. As we will see in Section 4, this modification improves the precision of the extracted instances, especially in the early stages of iteration.

3.2 Scaling Factor in Reliability Scores

Another modification to the *Espresso* algorithm to reduce the power of generic patterns is to use *local* $\max_{pmi}$ instead of *global* $\max_{pmi}$. Since PMI ranges $[-\infty, +\infty]$, the point of dividing $pmi(i,p)$ by $\max_{pmi}$ in *Espresso* is to normalize the reliability to [0, 1]. However, using PMI directly to estimate the reliability of a pattern when calculating the reliability of an instance may lead to unexpected results because the absolute value of PMI is highly variable across instances and patterns. We define the *local* $\max_{pmi}$ of the reliability of an instance to be the absolute value of the maximum PMI *for a given instance*, as opposed to taking the maximum for all instances in a given iteration. Local $\max_{pmi}$ of the reliability of a pattern is defined in the same way. As we show in the next section, this modification has a large impact on the effectiveness of our algorithm.

3.3 Performance Improvements

Tchai, unlike *Espresso*, does not perform the pattern induction step between iterations; rather, it simply recomputes the reliability of the patterns induced at the beginning. Our assumption is that fairly reliable patterns will occur with at least one of the seed instances if they occur frequently enough in query logs. Since pattern induction is computationally expensive, this modification reduces the computation time by a factor of 400.

Category	Seeds (with English translation)
Travel	jal, ana, jr, じゃらん(jalan), his
Finance	みずほ銀行(Mizuho Bank), 三井住友銀行 (SMBC), jcb, 新生銀行 (Shinsei Bank), 野村證券(Nomura Securities)

Table 2: Seed instances for Travel and Financial Services categories

4 Experiment

In this section, we present an empirical comparison of *Tchai* with the systems described in Section 2.

4.1 Experimental Setup

Query logs: The data source for instance extraction is an anonymized collection of query logs submitted to Live Search from January to February 2007, taking the top 1 million unique queries. Queries with garbage characters are removed. Almost all queries are in Japanese, and are accompanied by their frequency within the logs.

Target categories: Our task is to learn word categories that closely reflect the interest of web search users. We believe that a useful categorization of words is task-specific, therefore we did not start with any externally available ontology, but chose to start with a small number of seed words. For our task, we were given a list of 23 categories relevant for web search, with a manual classification of the 10,000 most frequent search words in the log of December 2006 (which we henceforth refer to as the *10K list*) into one of these categories.[2] For evaluation, we chose two of the categories, Travel and Financial Services: Travel is the largest category containing 712 words of the 10K list (as all the location names are classified into this category), while Financial Services was the smallest, containing 240 words.

Systems: We compared three different systems described in Section 2 that implement an iterative algorithm for lexical learning:

[2] The manual classification assigns only one category per word, which is not optimal given how ambiguous the category memberships are. However, it is also very difficult to reliably perform a multi-class categorization by hand.

	10K list		Not in 10K list
	Travel	Not Travel	
Travel	280	17	251
Not Travel	0	7	125

Table 3: Comparison with manual annotation: Travel category

	10K list		Not in 10K list
	Finance	Not Finance	
Finance	41	30	30
Not Finance	0	5	99

Table 4: Comparison with manual annotation: Financial Services category

- *Basilisk*: The algorithm by (Thelen and Riloff, 2002) described in Section 2.
- *Espresso*: The algorithm by (Pantel and Pennacchiotti, 2006) described in Sections 2 and 3.
- *Tchai*: The *Tchai* algorithm described in this paper.

For each system, we gave the same seed instances. The seed instances are the 5 most frequent words belonging to these categories in the 10K list; they are given in Table 2. For the Travel category, "jal" and "ana" are airline companies, "jr" stand for Japan Railways, "jalan" is an online travel information site, and "his" is a travel agency. In the Finance category, three of them are banks, and the other two are a securities company and a credit card firm. *Basilisk* starts by extracting 20 patterns, and adds 100 instances per iteration. *Espresso* and *Tchai* start by extracting 5 patterns and add 200 instances per iteration. *Basilisk* and *Tchai* iterated 20 times, while *Espresso* iterated only 5 times due to computation time.

4.2 Results

4.2.1 Results of the *Tchai* algorithm

Tables 3 and 4 are the results of the *Tchai* algorithm compared to the manual classification. Table 3 shows the results for the Travel category. The precision of *Tchai* is very high: out of the 297 words classified into the Travel domain that were also in the 10K list, 280 (92.1%) were learned rectly.[3] It turned out that the 17 instances that represent the precision error were due to the ambiguity of hand labeling, as in 東京ディズニーランド 'Tokyo Disneyland', which is a popular travel destination, but is classified as Entertainment in the manual annotation. We were also able to correctly learn 251 words that were not in the 10K list according to manual verification; we also harvested 125 new words "incorrectly" into the Travel domain, but these words include common nouns related to Travel, such as 釣り 'fishing' and レンタカー 'rental car'. Results for the Finance domain show a similar trend, but fewer instances are extracted.

Sample instances harvested by our algorithm are given in Table 5. It includes subclasses of travel-related terms, for some of which no seed words were given (such as Hotels and Attractions). We also note that segmentation errors are entirely absent from the collected terms, demonstrating that query logs are in fact excellently suited for acquiring new words for languages with no explicit word segmentation in text.

Type	Examples (with translation)
Place	トルコ (Turkey), ラスベガス (Las Vegas), バリ島 (Bali Island)
Travel agency	Jtb, トクー (www.tocoo.jp), yahoo (Yahoo ! Travel), net cruiser
Attraction	ディズニーランド (Disneyland), usj (Universal Studio Japan)
Hotel	帝国ホテル(Imperial Hotel), リッツ(Ritz Hotel)
Transportation	京浜急行(Keihin Express), 奈良交通(Nara Kotsu Bus Lines)

Table 5: Extracted Instances

4.2.2 Comparison with *Basilisk* and *Espresso*

Figures 1 and 2 show the precision results comparing *Tchai* with *Basilisk* and *Espresso* for the Travel and Finance categories. *Tchai* outperforms *Basilisk* and *Espresso* for both categories: its precision is constantly higher for the Travel category, and it achieves excellent precision for the Finance category, especially in early iterations. The differences in behavior between these two categories are due to the inherent size of these domains. For the

[3] As the 10K list contained 712 words in the Travel category, the recall against that list is fairly low (~40%). The primary reason for this is that all location names are classified as Travel in the 10K list, and 20 iterations are not enough to enumerate all frequent location names. Another reason is that the 10K list consists of *queries* but our algorithm extracts *instances* – this sometimes causes a mismatch, e.g.,*Tchai* extracts リッツ 'Ritz' but the 10K list contains リッツホテル 'Ritz Hotel'.

	# of inst.	Precision	Rel.recall
Basilisk	651	63.4	1.26
Espresso	500	65.6	1.00
Tchai	680	80.6	1.67

Table 6: Precision (%) and relative recall: Travel domain

	# of inst.	Precision	Rel.recall
Basilisk	278	27.3	0.70
Espresso	704	15.2	1.00
Tchai	223	35.0	0.73

Table 7: Precision (%) and relative recall: Financial Services domain

smaller Finance category, *Basilisk* and *Espresso* both suffered from the effect of generic patterns such as #ホームページ 'homepage' and #カード 'card' in early iterations, whereas *Tchai* did not select these patterns.

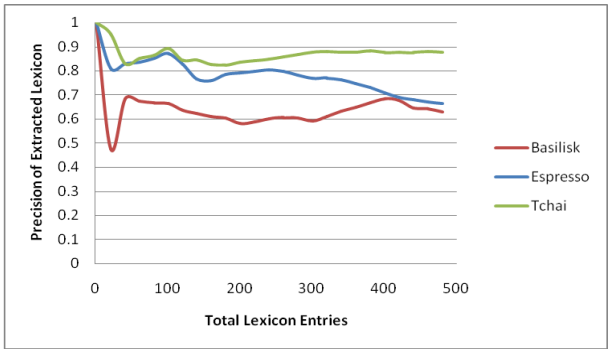

Figure 1: *Basilisk*, *Espresso* vs. *Tchai*: Travel

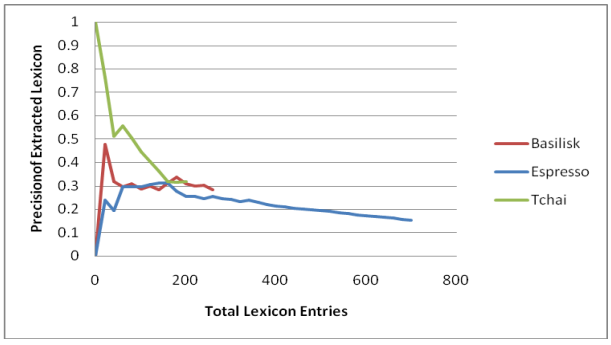

Figure 2: *Basilisk*, *Espresso* vs. *Tchai*: Finance

Comparing these algorithms in terms of recall is more difficult, as the complete set of words for each category is not known. However, we can estimate the *relative* recall given the recall of another system. Pantel and Ravichandran (2004) defined *relative recall* as:

$$R_{A|B} = \frac{R_A}{R_B} = \frac{C_A/C}{C_B/C} = \frac{C_A}{C_B} = \frac{P_A \times |A|}{P_B \times |B|}$$

where $R_{A|B}$ is the relative recall of system A given system B, C_A and C_B are the number of correct instances of each system, and C is the number of true correct instances. C_A and C_B can be calculated by using the precision, P_A and P_B, and the number of instances from each system. Using this formula, we estimated the relative recall of each system relative to *Espresso*. Tables 6 and 7 show that *Tchai* achieved the best results in both precision and relative recall in the Travel domain. In the Finance domain, *Espresso* received the highest relative call but the lowest precision. This is because *Tchai* uses a filtering method so as not to select generic patterns and instances.

Table 8 shows the context patterns acquired by different systems after 4 iterations for the Travel domain.[4] The patterns extracted by *Basilisk* are not entirely characteristic of the Travel category. For example, "*p#sonic*" and "*google+#lytics*" only match the seed word "ana", and are clearly irrelevant to the domain. *Basilisk* uses token count to estimate the score of a pattern, which may explain the extraction of these patterns. Both *Basilisk* and *Espresso* identify location names as context patterns (e.g., #東京 'Tokyo', #九州 'Kyushu'), which may be too generic to be characteristic of the domain. In contrast, *Tchai* finds context patterns that are highly characteristic, including terms related to transportation (#+格安航空券 'discount plane ticket', #マイレージ 'mileage') and accommodation (#+ホテル 'hotel').

4.2.3 Contributions of *Tchai* components

In this subsection, we examine the contribution of each modification to the *Espresso* algorithm we made in *Tchai*.

Figure 3 illustrates the effect of each modification proposed for the *Tchai* algorithm in Section 3 on the Travel category. Each line in the graph corresponds to the *Tchai* algorithm with and without the modification described in Sections 3.1 and 3.2. It shows that the modification to the $\max_{pmi}$ function (purple) contributes most significantly to the improved accuracy of our system. The filtering of generic patterns (green) does not show

[4] Note that *Basilisk* and *Espresso* use context patterns only for the sake of collecting instances, and are not interested in the patterns per se. However, they can be quite useful in characterizing the semantic categories they are acquired for, so we chose to compare them here.

System	Sample Patterns (with English translation)
Basilisk	#東日本(*east_japan*), #西日本(*west_japan*), *p#sonic*, #時刻表(*timetable*), #九州(*Kyushu*), #+マイレージ(*mileage*), #バス(*bus*), *google+#lytics*, #+料金(*fare*), #+国内(*domestic*), #ホテル(*hotel*)
Espresso	#バス(*bus*), 日本#(*Japan*), #ホテル(*hotel*), #道路(*road*), #イン(*inn*), フジ#(*Fuji*), #東京(*Tokyo*), #料金(*fare*), #九州(*Kyushu*), #時刻表(*timetable*), #+旅行(*travel*), #+名古屋(*Nagoya*)
Tchai	#+ホテル(*hotel*), #+ツアー(*tour*), #+旅行(*travel*), #予約(*reserve*), #+航空券(*flight_ticket*), #+格安航空券(*discount_flight_titcket*), #マイレージ(*mileage*), 羽田空港+#(*Haneda Airport*)

Table 8: Sample patterns acquired by three algorithms

a large effect in the precision of the acquired instances for this category, but produces steadily better results than the system without it.

Figure 4 compares the original *Espresso* algorithm and the modified *Espresso* algorithm which performs the pattern induction step only at the beginning of the bootstrapping process, as described in Section 3.3. Although there is no significant difference in precision between the two systems, this modification greatly improves the computation time and enables efficient extraction of instances. We believe that our choice of the seed instances to be the most frequent words in the category produces sufficient patterns for extracting new instances.

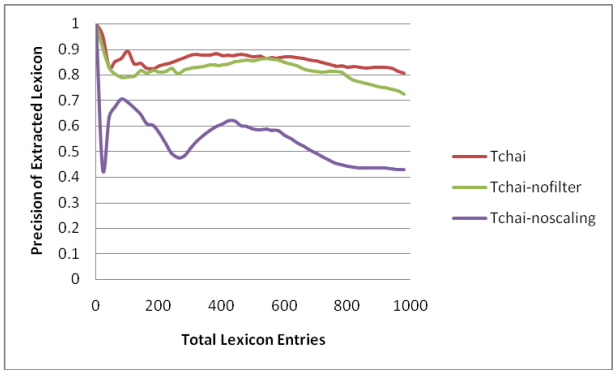

Figure 3: System precision w/o each modification

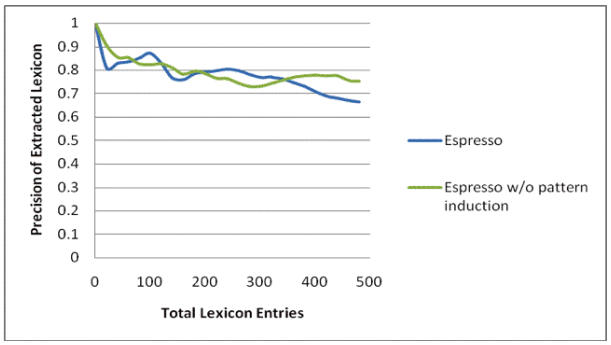

Figure 4: Modification to the pattern induction step

5 Conclusion

We proposed a minimally supervised bootstrapping algorithm called *Tchai*. The main contribution of the paper is to adapt the general-purpose *Espresso* algorithm to work well on the task of learning semantic categories of words from query logs. The proposed method not only has a superior performance in the precision of the acquired words into semantic categories, but is faster and collects more meaningful context patterns for characterizing the categories than the unmodified *Espresso* algorithm. We have also shown that the proposed method requires no pre-segmentation of the source text for the purpose of knowledge acquisition.

Acknowledgements

This research was conducted during the first author's internship at Microsoft Research. We would like to thank the colleagues at Microsoft Research, especially Dmitriy Belenko and Christian König, for their help in conducting this research.

References

Sergey Brin. 1998. Extracting Patterns and Relations from the World Wide Web. WebDB Workshop at 6th International Conference on Extending Database Technology, EDBT '98. pp. 172-183.

Michael Collins and Yoram Singer. 1999. Unsupervised Models for Named Entity Classification. *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*. pp. 100-110.

Oren Etzioni, Michael Cafarella, Dong Downey, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S. Weld, and Alexander Yates. 2005. Unsupervised Named-Entity Extraction from the Web: An Experimental Study. *Artificial Intelligence*. 165(1). pp. 91-134.

Marti Hearst. 1992. Automatic Acquisition of Hyponyms from Large Text Corpora. *Proceedings of the*

Fourteenth International Conference on Computational Linguistics. pp 539-545.

Patrick Pantel and Marco Pennacchiotti. 2006. Espresso: Leveraging Generic Patterns for Automatically Harvesting Semantic Relations. *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the ACL*. pp. 113-120.

Patrick Pantel and Deepak Ravichandran. 2004. Automatically Labeling Semantic Classes. *Proceedings of Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT/NAACL-04)*. pp. 321-328.

Marius Paşca. 2004. Acquisition of Categorized Named Entities for Web Search. *Proceedings of the 13th ACM Conference on Information and Knowledge Management (CIKM-04)*. pp. 137-145.

Marius Paşca. 2007. Organizing and Searching the World Wide Web of Fact – Step Two: Harnessing the Wisdom of the Crowds. *Proceedings of the 16th International World Wide Web Conference (WWW-07)*. pp. 101-110.

Marius Paşca and Benjamin Van Durme. 2007. What You Seek is What You Get: Extraction of Class Attributes from Query Logs. *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*. pp. 2832-2837.

Marius Paşca, Dekang Lin, Jeffrey Bigham, Andrei Lifchits and Alpa Jain. 2006. Organizing and Searching the World Wide Web of Facts – Step One: the One-Million Fact Extraction Challenge. *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI-06)*. pp. 1400-1405.

Ellen Riloff. 1996. Automatically Generating Extraction Patterns from Untagged Text. *Proceedings of the Thirteenth National Conference on Artificial Intelligence*. pp. 1044-1049.

Ellen Riloff and Rosie Jones. 1999. Learning Dictionaries for Information Extraction by Multi-Level Bootstrapping. *Proceedings of the Sixteenth National Conference on Artificial Intellligence (AAAI-99)*. pp. 474-479.

Satoshi Sekine and Hisami Suzuki. 2007. Acquiring Ontological Knowledge from Query Logs. *Proceedings of the 16th international conference on World Wide Web*. pp. 1223-1224.

Craig Silverstein, Monika Henzinger, Hannes Marais, and Michael Moricz. 1998. Analysis of a Very Large AltaVista Query Log. *Digital SRC Technical Note* #1998-014.

Michael Thelen and Ellen Riloff. 2002. A Bootstrapping Method for Learning Semantic Lexicons using Extraction Pattern Contexts. *Proceedings of Conference on Empirical Methods in Natural Language Processing*. pp. 214-221.