Bootstrap Pattern Learning for Open-Domain CLQA

Hideki Shima
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA, 15213 USA
hideki@cs.cmu.edu

Teruko Mitamura
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA, 15213 USA
teruko@cs.cmu.edu

ABSTRACT

We describe Javelin, a Cross-lingual Question Answering system which participated in the NTCIR-8 ACLIA evaluation and which is designed to work on any type of question, including factoid and complex questions.

The key technical contribution of this paper is a minimally supervised bootstrapping approach to generating lexico-syntactic patterns used for answer extraction. The preliminary evaluation result (measured by nugget F3 score) shows that the proposed pattern learning approach outperformed two baselines, a supervised learning approach used in NTCIR-7 ACLIA and a simple key-term based approach, for both monolingual and crosslingual tracks. The proposed approach is general and thus it has potential applicability to a wide variety of information access applications which require deeper semantic processing.

Keywords

Pattern learning, minimally-supervised learning, bootstrapping.

1. INTRODUCTION

LTI's Javelin is a Cross-lingual Question Answering (QA) system for any type, of question, including factoid and complex questions. Javelin has a pipeline architecture which consists of four main modules:

- **Question Analyzer:** Responsible for analyzing the question to determine the information need (question type, answer type, key terms, etc.).
- **Retrieval Strategist (RS):** Responsible for extracting a ranked list of answer-bearing documents, using a query formulated using information provided by the Question Analyzer.
- **Information eXtractor (IX):** Responsible for extracting and scoring/ranking answer candidates from the answer bearing documents.
- **Answer Generator (AG):** Responsible for removing duplicates and selecting/filtering answers.

All the modules are designed to be language independent, and utilize uniform interfaces to MT and NLP services to support run-time loading of language-specific resources. This paper mainly focuses on the IX module for answer extraction where the key technical contribution has been made. To find more details for other modules, see [13][14][25]. For the IR4QA task, we used the Question Analyzer and RS modules. For the CCLQA task, we take retrieved documents from the IR4QA task, and subsequently run the IX and AG on them.

2. IR4QA

In this section, we describe our system for the IR4QA task [21]. The system basically works as follows. Given the question, the Question Analyzer identifies the key terms used for retrieval, then a Google translator translates the keyterms for the EN-JA subtask.

Key term extraction The key term extractor is responsible for creating a list of terms that will be useful for both retrieving potentially relevant answer-bearing documents and subsequently extracting answers from those documents. Using NLP tools, the key term extractor identifies a set of noun phrases, which is extended with any named entities (NEs) that were recognized.

Because the tasks we participated in are not just ad hoc retrieval tasks, any retrieval errors will affect the performance of later modules. Thus we decided to design the RS module to favor recall over precision, using the Indri retrieval engine (language model + inference network) with fail-safe query formulation and a character-based index.

Fail safe query formulation Basically, the strategy is to give credits to queries from different approaches. In Indri query language, queries look like:

- `#weight(10 EXACT_PHRASE 1 PARTIAL_PHRASE)`
- `#weight(10 KEY_TERMS 1ALL_TERMS_IN_QUESTION)`

In this way, we can retrieve passages even when all key terms aren't found.

Character based index Use of character-based indexing is based on our error analysis of the word-level indexing approach at NTCIR-7 ACLIA IR4QA; we found that morphological boundary detection errors cause a mismatch between the query and relevant documents, especially for NEs.

Evaluation Result We submitted three kinds of runs for each monolingual and crosslingual task, where 03 runs are the simple baselines with key phrases only, 02 runs used key phrases (exact match) relaxed with key terms (partial match), and 01 runs used fail-safe query formulation where key phrases are relaxed with question sentence terms. The result is shown in Table 1.

Table 1. IR4QA run results (the revised version after the bugfix [21])

Run ID	Average Precision
LTI-EN-JA-01-T	0.3327
LTI-EN-JA-02-T	0.3293
LTI-EN-JA-03-T	0.3074
LTI-JA-JA-01-T	0.4356
LTI-JA-JA-02-T	0.4351
LTI-JA-JA-03-T	0.4293

3. CCLQA

In this section, we describe our system for the CCLQA task [15], centering the focus on the answer extraction module. Answer extraction is one of the core tasks in a question answering system where the goal is to identify answer candidates from retrieved passages, and then rank them according to a confidence score. An ideal answer extractor would satisfy the following desiderata:

- **Coverage (recall):** It must extract as many correct answers as possible.
- **Reliability (precision)**: It must return correct responses with less noise.
- **Minimum human effort**: It must be implementable with minimum human effort, or with minimal/light supervision for learning-based algorithms.
- **Generality:** It must be applicable to various types of questions, whether complex or factoid.
- **Efficiency**: It must run efficiently in terms of speed, disk space, and memory usage at run time (highest priority) and batch training time.
- **Portability**: It must support inter-domain and inter-language portability.

The QA community has investigated several approaches to scoring answer-bearing passages and extracting answers. In addition to lightweight bag-of-words representations and term proximity based method, strategies can be categorized as follows.

- Pattern based approaches [8][20][22][23][28]
- Sentence level similarity approach (between question and answer-bearing-passage) based on:
  - Syntactic structures [3][19][24]
  - Semantic structures [17][26]
  - Statistical machine-translation-inspired models [5][16][27]
  - Textual entailment based model [7]

Sentence similarity approaches are theoretically sound, but have two drawbacks in relation to our task. Firstly, these approaches assume that question and answer-bearing passages share similar or identical syntactic and semantic structures. This assumption often holds for factoid questions, but does not hold for many complex questions which have linguistically simple representations such as “What is X?” or “Who is X?”. Secondly, there is a practical issue in that the recall and precision of the parsers used in sentence similarity approaches may not be good enough to outperform simpler approaches. This led to the early observation that “linguistically-impoverished systems have generally outperformed those that attempt syntactic or semantic analysis” [9]. Errors are seen particularly in parsing of questions, due to lack of interrogative sentences in the parser’s training data. Parsing errors can result in a failure to capture answers even though answer-bearing documents can be obtained (coverage issue); tools, resources and large-scale tagged corpora aren’t necessarily available for all languages (portability issue); and machine learning based tools can take a lot of time in processing (efficiency issue).

One strategy that has been proven to work well on large scale evaluation uses lexico-syntactic patterns (hereafter LSP) or surface patterns [8]. It works in the following way: suppose the given question is “When was Basho Matuo born?”, an LSP “*x* was born in *y*” can be instantiated with the question term as “Basho Matsuo was born in *y*” and used to identify the answer *y* from the corpus.

An interesting research question is how to obtain useful LSPs with minimum human effort given training data (questions and gold standard answers) and unstructured text corpora Hand-crafted patterns are generally very accurate, but there is a coverage problem even when expanding expressions with thesaurus data, and manual pattern creation generally requires a substantial human effort. Given this background, an automatic approach to generating patterns for QA is one of the active research areas in the QA community, with promising empirical results for large scale collections [20][22][28].

In this section, we present a minimally-supervised bootstrap pattern learning algorithm with unique contributions to improve coverage, reliability, batch-time efficiency, and inter-language portability. In the rest of this section, we introduce our base work on bootstrap learning in Subsection 4.1. Then, in Subsection 4.2, we describe our method, addressing previously unsolved issues. Finally we present the formal run settings and results in Subsection 4.3.

3.1 Bootstrap Learning

We will briefly introduce a minimally-supervised bootstrap learning framework called *Espresso* [18]. Basically, *Espresso* takes small number (~10) of seed instances (in case of binary relation learning, an instance is made of a pair of texts such as “Basho Matsuo” and “1644”) and generates patterns that capture instances. The algorithm is iterative; more instances are found with the patterns, and new patterns are generated from the newly found instances, and so on. Figure 1 visualizes the iterative learning process.

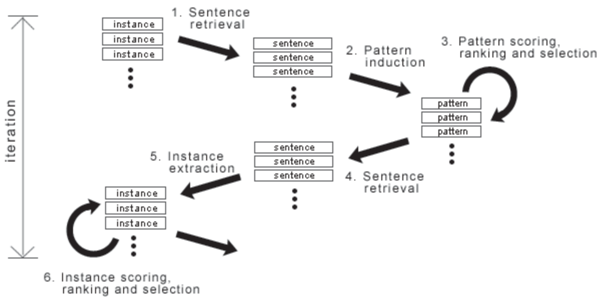

Figure 1. Bootstrapping pattern learning overview

More detailed steps are described below.

1. **Sentence retrieval**: Let *I* denote a set of instances. For each instance $i \in I$ consisting of two terms *x* and *y* (i.e i={*x*, *y*}), retrieve all sentences S_i each including the instance.
2. **Pattern Induction:** Extract substrings linking terms *x* and *y* from S_i to form a set of patterns *P*.
3. **Reliability calculation on patterns**: Score *P* according to reliability scores based on approximated PMI statistics using the formula (1) below, and adopt only “reliable” patterns
4. **Instance generation**: Extract new instances *I’* using adopted *P*.
5. **Reliability calculation on instances**: Likewise in step 3, calculate reliability scores for all instances in *I’* using the formula (2) below, and select only “reliable” instances.

(Repeat step 1-5 until convergence).

$$r_\pi(p) = \frac{\sum_{i \in I} \left(\frac{pmi(i,p)}{\max_{pmi}} * r_\iota(i) \right)}{|I|} \qquad (1)$$

$$r_\iota(p) = \frac{\sum_{p \in P'} \left(\frac{pmi(i,p)}{\max_{pmi}} * r_\pi(p) \right)}{|P|} \qquad (2)$$

Note the symmetry in (1) and (2), where previous instance reliabilities are used to calculate pattern reliabilities, and vice versa. The reliability score is 1 for the seed instances.

Note that the goal of most previous works with bootstrapping method [1][2][6][10] is to acquire instances from the seed, and patterns themselves are treated as a subsidiary outcome. In our task, we aim to acquire output patterns and discard the instances found.

3.2 Proposed pattern learning approach

We hypothesize that end-to-end QA metric scores improve if we can improve the pattern learning algorithm for answer extraction. By improving the algorithm, we mean to support minimal supervision to learn more from less resources, in order to realize better coverage and reliability by overcoming previously unaddressed issues.

In the rest of this subsection, we will describe specific details in 4.2.1 to 4.2.3, and practical implementation details in 4.2.4.

3.2.1 Sentence generation

In the pattern induction phase, we would like to find patterns *P*, which can capture instance term(s), given instances *I*. The algorithm we adopted to find patterns among S_i is called Longest Common Substring, or *LCSubstr* which finds the longest consecutive strings. For instance, let's assume we retrieve sentences consisting of words $s_1 = \{w_1, w_2, x, w_3, w_4, y, w_5, w_6\}$ and $s_2 = \{w_7, x, w_3, w_4, y, w_5, w_8\}$. The pattern induced from s_1 and s_2 is $\{x, w_3, w_4, y, w_5\}$, which is the longest consecutive words including $i=\{x,y\}$.

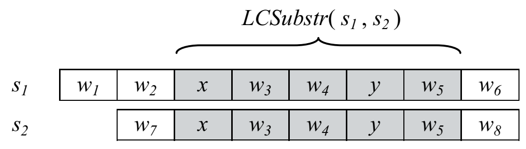

Figure 2. *LCSubstr* example

In order to have better opportunities for *LCSubstr* to work, sentence generalization is important. And it is especially important for languages such as Japanese where word order is relatively free[1]. Free word order makes the learning task more difficult, as we observe fewer matching substrings. To mitigate this data sparseness problem, we introduce two generalization techniques: one by finer-grained labeling with NEs, and another by rule-based constituent removal.

Generalization by NE labels

Retrieved sentences contain terminology which is too specific to generate general patterns. *Espresso* proposes to replace terminological expressions with the *TR* label (e.g. In chemical domain, a sentence "Because HF is a weak acid and *x* is a *y*" is generalized into "Because *TR* is a *TR* and *x* is a *y*"), however, as Pantel and Pennacchiotti admit, generalized patterns are less precise [18]. Given that, we would like to introduce sentence generalization using NE labels, which represent an intermediate labeling somewhere between surface text and a more general TR label.

The generalization process proposed here can be done with any NE recognizer. A sample from S_i and from generalized sentences S'_i are shown below in (1a) and (1b) respectively with generalized strings underlined.

(1)(a) 現代自動車（韓国）の日本法人「<NP2>」（千葉県印西市、<NP1>社長）は２５日、…

(1)(b) <COMPANY>（<COUNTRY>）の日本法人「<NP2>」（<PLACE>、<NP1>社長）は<DATE>、…

The patterns are induced from the sentences $S_i \cup S'_i$.

Generalization by adjunct phrase removal

We remove adjunct phrases, or phrases whose removal does not affect grammatical well-formedness of the sentence. More specifically, we remove consecutive strings starting with a kanji character and ending with a particle, except for the subject case marking particle GA and the topic particle WA. In the example below, underlined strings in (2a) match the adjunct phrase rules mentioned above, and are removed.

(2)(a) <NP1>博士が米国滞在中に著した英文本「<NP2>～日本の心」が …

(2)(b) <NP1>博士が著した英文本「<NP2>～日本の心」が …

Following generalization, sentences $S_i \cup S' \cup S''_i$ are used as a source for inducing patterns.

3.2.2 Leveraging non-word tokens

Non-word tokens (e.g. comma, parentheses, quotation marks etc), play an important role, although previous work did not fully address them. Incorporating non-words in patterns is important when the target text (e.g., newswire articles) often condenses information using symbols to attain conciseness. Coverage can be enhanced if we can properly include non-word tokens in LSPs, which parsing-based approaches cannot easily accommodate (since they abstract away from surface punctuation, for the most part). Another motivation to use non-word information is to identify the proper boundary of the instance to extract; e.g. book titles are usually very difficult to identify, but if quotation symbols are included in LSPs we can better detect the title boundaries.

There is a practical challenge in learning patterns with non-word tokens in *Espresso*. As a part of processing to estimate pattern reliability based on PMI, which is potentially the most time consuming part of the learning, we need to count how many times a pattern appears in the corpus. However, this is very challenging to do quickly, partly because standard search engine indexes do not contain symbol characters; a crude approach of counting expressions with regular expression matching could require a computationally intractable amount of time. In 0, we describe the solution to this challenge in detail.

Example LSPs learned with the proposed method for the AUTHOR relationship are exemplified below. Notice non-word

[1] SOV is the canonical order, but OSV is also often seen.

symbol tokens such as quotation characters ‘「’, ‘」’, ‘『’, ‘』’, comma ‘、’, interpunct ‘・’ and bullet ‘◇’, which add informative context to the pattern, as well as indicating the variable boundary.

```
0.012 <NP1>著「<NP2>」
0.009 <NP1>さんのエッセー「<NP2>」
0.008 「<NP2>」の作者、<NP1>の
0.008 俳聖・<NP1>が「<NP2>
0.008 <NP1>の「<NP2>」の一節
0.007 ベストセラー「<NP2>」の作者、<NP1>
0.005 <NP1>の小説「<NP2>」
0.005 ◇<NP1>著「<NP2>」
0.005 <NP1>の『<NP2>』（
```

Figure 3. AUTHOR patterns learned with the proposed algorithm

3.2.3 Partial use of generic patterns

Consider a *generic* pattern “*x* by *y*”. Since it is very general, there is not enough context to restrict the pattern to represent a specific relationship. For instance, it can capture an instance which consists of a movie name and its director, when what we want to find is a book name and its author. Reliability of generic patterns may be low as they can capture irrelevant instances; however, if they can be used in a smart way, we can expect much better coverage and added confidence in the correct instances. To this end, the use of generic patterns is a very interesting research topic in pattern learning; Pantel and Pennacchiotti [18] took advantage of generic patterns; other works have investigated the “semantic drift” phenomena caused by the use of generic patterns [4][11].

Pantel and Pennacchiotti reports two experimental settings with *Espresso*, one called *ESP-* where generic patterns are completely filtered out and another called *ESP+* where all patterns are used, including generic ones. Experimental results on instance learning show that *ESP-* achieves high precision and low recall, and *ESP+* achieves high recall and low precision. Low recall is problematic in our task because the low likelihood that patterns will fire in answer extraction implies a minimal impact on end-to-end results. From that practical point of view, a middle ground solution between these two extremes is desirable. Given this background, we avoided using generic patterns in instance extraction, but did use them for calculating instance reliability and include them in the final pattern list.

3.2.4 Engineering issues in practice

There are many technical hurdles that prevent one from implementing the proposed pattern learner. For instance, it is not straightforward to retrieve sentences that contain a birthplace pattern “*x* (<YEAR> -) was born in *y*”, or count the occurrences in a corpus using a standard search engine. We recommend Indri[2] for its capability of retrieving and counting passages which match queries containing annotations and symbols, more details are provided below.

Why use Indri

The PMI calculation is one of the most important and time consuming components in the proposed algorithm. The PMI scores between two events are calculated in the following way.

$$pmi(x, y) = \log \frac{P(x, y)}{P(x)P(y)}$$

When we calculate PMI between instance and pattern, the traditional approach approximates it with document hits.

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

Suppose we have N instances to score M candidate prototypes, hits for N * M + N + M. In a realistic scenario where N=200, M=5000, the total number of hit calculations is over 200 * 500 = 1 million per iteration[3]. Suppose the learning converges in 5 iterations, and we have 20 relations to learn; then the total number of hit calculations is 1 million * 5 * 20 = 100 million. Given that, we cannot use an expensive technique to count expressions (such as using *grep* or regular expression matching). Instead of using these slower legacy approaches, fast computation with Indri is indispensable for the learning task to be computationally tractable.

How to index a corpus with NE/sentence annotation

One can index annotations with Indri off-the-shelf. Here are steps with examples in Indri 2.11.

1. Format the corpus in TREC format, and encode the text in UTF-8. You can include extra tags between `<DOCNO>` and `<TEXT>`.
2. For Japanese, Chinese, or any languages where the writing system does not have word boundaries with spaces, tokenize text with spaces. For example, in case of morpheme based indexing, insert spaces between morphemes.
3. Create annotations. You can either insert annotation tags inline, or create a stand-off annotation file as shown in the second row in Figure 4.
4. We recommend stand-off annotation as it can represent overlapping annotations. Each line corresponds with an annotation containing DOCNO, type (TAG), annotation id, annotation name, annotation begin position in byte offset, annotation length in bytes, optional field for TAG, parent annotation ID for structured retrieval, and any text for debug purpose.
5. Create an index. In such a way as
   ```
   $ ./bin/buildindex parameter.txt
   ```
6. Verify the index with `dumpindex` or `runquery` tool.

[2] http://www.lemurproject.org/indri/

[3] Some portion of hit count can be reused in cache for later iterations though

corpus.txt
```
<DOC>
  <DOCNO>DOC001</DOCNO>
  <TEXT>
 江戸時代の俳人、松尾芭蕉（１６４４～
９４年）の代表作「奥の細道」の
   : : :
  </TEXT>
</DOC>
<DOC>
: : :
```

annotation.txt
```
D001 TAG 1 SENTENCE  41  420 0 0
D001 TAG 2 PERSON    65  15  0 1
D001 TAG 3 DATE      117 15  0 1
D001 TAG 4 DATE      137 11  0 1
```

parameter.txt
```
<parameters>
  <memory>1g</memory>
  <index>corpus.index</index>
  <corpus>
    <path>corpus.txt</path>
    <annotations>annotation.txt</annotations>
    <class>trectext</class>
  </corpus>
  <metadata><forward>docno</forward></metadata>
  <field><name>sentence</name></field>
  <field><name>person</name></field>
  <field><name>date</name></field>
   : : :
</parameters>
```

Figure 4. Sample corpus in TREC format, offset annotation, and parameter file for buildindex

How to index non-word tokens

In order to index non-word tokens, one needs to modify the source code `src/UTF8Transcoder.cpp`. In the source code, there are character categories declared in hexadecimal notation. For each non-word tokens to be indexed, for example:

```
{0xFE10, 0xFE19, indri::parse::CharClass::punctuation}
```

Specify the CharClass to be letter such as:

```
{0xFE10, 0xFE19, indri::parse::CharClass::letter}
```

After rebuilding the code, you will be able to treat non-word tokens as letters.

Query formulation

To count how many times a pattern appears in the corpus, use the "expression count" option in `dumpindex` tool. Suppose we calculate a reliability score of a pattern:

```
<NP1>は<YEAR>に<NP2>で誕生した
```

We can calculate the number of times the pattern occurs within a certain span, using an ordered window query *#odN* and a typed wild card query *#any*:

```
#od30( #od1( は #any:year に)
       #od1( で 誕 生 し た ) )
```

In order to retrieve sentences containing two terms "松尾芭蕉" and "１６４４年", query with an extent restriction with a field *sentence* and an unordered window query *#uw*:

```
#combine[sentence]( #uw(
  #od1( 松 尾 芭 蕉 ) #od1( １ ６ ４ ４ 年 ) ) )
```

3.3 Evaluation Result

In this subsection, we present experimental settings and the formal evaluation result on NTCIR-8 ACLIA CCLQA task.

Seed generation We hand-crafted 10 to 20 seed instances where binary-argument pairs of question term and answer. As a result of batch time training, we obtained LSPs such as the examples shown in Table 2.

Answering Factoid and complex questions Answers to complex questions must satisfy a complex information need. As inspired by the complex question decomposition method by Lacatusu et al [12], one can see a complex question as a set of factoid questions. For example, a complex question "Who is X" can be decomposed into "When was X born?", "What award did X receive?" etc. Using this analogy, we used multiple factoid patterns to answer a complex question.

Table 2. Sample of seed instances, or pairs of award recipient and award name, and patterns actually learned.

Seeds	Learned patterns
(アンジェイ・ワイダ，特別名誉賞)	<NP2>受賞者の<NP1>
(山田五十鈴，文化勲章)	<NP2>を受賞した<NP1>の
(松井秀喜，ゴールデンスピリット賞)	「<NP2>」は<NP1>が
(片岡篤史，ゴールデンスピリット賞)	<NP2>は映画監督の<NP1>氏以来
(ロバート・カーン，国家技術メダル)	故<NP1>氏に対する<NP2>表彰式。
(中村修二，仁科記念賞)	、<NP2>受賞者の<NP1>
(江崎玲於奈，ノーベル物理学賞)	<NP2>は<NP1>氏が加わり
(田中澄江，読売文学賞)	純文学対象の<NP2>が<NP1>さん（
(黒沢明，国民栄誉賞)	<NP2>を受賞した<NP1>
(村上龍，芥川賞)	で<NP2>受賞者の<NP1>
(ギュンター・グラス，ノーベル文学賞)	<NP2>受賞者、<NP1>

As a source for extracting answers, we used the Mainichi corpus only. As a baseline, we used the NTCIR-7 algorithm in 01 runs and a simple key term overlap approach in 03 runs to compare against the 02 runs with the answer extraction module using LSPs learned in bootstrapping. The following table shows the summary of results where the proposed approach outperforms baseline runs in both EN-JA and JA-JA.

Table 3. CCLQA run results (preliminary) [15]

Run ID	Nugget F3
LTI-EN-JA-01-T	0.1074
LTI-EN-JA-02-T	0.1130
LTI-EN-JA-03-T	0.1045
LTI-JA-JA-01-T	0.1069
LTI-JA-JA-02-T	0.1443
LTI-JA-JA-03-T	0.1438

4. DISCUSSION AND CONCLUSION

The proposed approach is general enough that it has potential applicability to wide variety of information access applications. For instance, a set of patterns learned for each relationship can be used as templates for textual entailment or paraphrasing detection tasks. For further applications, the work can be used in automatic QA evaluation, where patterns are used to identify nuggets and system responses where surface texts differ but have equivalent meaning.

In this work, we proposed to use NEs to replace specific terms with an abstract label for increasing opportunities to learn more with smaller resource. Another direction of using NE Recognizer would be to restrict instance anchor terms with a NE category in order to add more constraint on patterns with a hope to improve reliability.

To conclude, we presented the LTI system participated in NTCIR-8 ACLIA where we addressed monolingual and crosslingual QA challenge on any types of questions.

As the key technical contribution of the paper, we described a minimally supervised bootstrapping approach to generating LSPs for answer extraction. The preliminary evaluation result (measured by nugget F3 score) shows that the proposed approach outperformed two baselines in both monolingual and crosslingual tasks. The proposed approach is general and thus it has potential applicability to wide variety of information access applications which require deeper semantic processing.

5. REFERENCES

[1] Bellare, Kedar, Partha Pratim Talukdar, Giridhar Kumaran, Fernando Pereira, Mark Liberman, Andrew McCallum, and Mark Dredze. 2008. Lightly-Supervised Attribute Extraction for Web Search. In Proceedings of NIPS 2007 Workshop on Machine Learning for Web Search.

[2] Carlson, Andrew, Justin Betteridge, Richard C. Wang, and Estevam R. Hruschka Jr. and Tom M. Mitchell. 2010. Coupled Semi-Supervised Learning for Information Extraction. In Proceedings of WSDM 2010.

[3] Cui, Hang, Renxu Sun, Keya Li, Min-Yen Kan, and Tat-Seng Chua. 2005. Question Answering Passage Retrieval using dependency relations. In Proceedings of SIGIR 2005.

[4] Curran, James R., Tara Murphy, and Bernhard Scholz. 2007. Minimising Semantic Drift with Mutual Exclusion Bootstrapping. In Proceedings of PACLING 2007.

[5] Echihabi, Abdessamad and Daniel Marcu. 2003. A Noisy-channel Approach to Question Answering. In Proceedings of ACL 2003.

[6] Hagiwara, Masato, Yasuhiro Ogawa, and Katsuhiko Toyama. 2009. Bootstrapping-based Extraction of Dictionary Terms from Unsegmented Legal Text. New Frontiers in Artificial Intelligence: JSAI 2008 Conference and Workshops, Revised Selected papers, Lecture Notes in Computer Science, Vol. 5447, pp. 213-227.

[7] Harabagiu, Sanda and Andrew Hickl. 2006. Methods for Using Textual Entailment in Open-Domain Question Answering. In Proceedings of COLING-ACL 2006.

[8] Jijkoun, Valentin, Maarten de Rijke, and Jori Mur. 2004. Information Extraction for Question Answering: Improving Recall Through Syntactic Patterns. In Proceedings of COLING 2004.

[9] Katz, Boris and Jimmy Lin. 2003. Selectively Using Relations to Improve Precision in Question Answering. In Proceedings of the EACL-2003 Workshop on Natural Language Processing for Question Answering.

[10] Komachi, Mamoru and Hisami Suzuki. 2008. Minimally Supervised Learning of Semantic Knowledge from Query Logs. In Proceedings of IJCNLP-08.

[11] Komachi, Mamoru, Taku Kudo, Masashi Shimbo, and Yuji Matsumoto. 2008. Graph-based Analysis of Semantic Drift in Espresso-like Bootstrapping Algorithms. In Proceedings of EMNLP 2008.

[12] Lacatusu, Finley, Andrew Hickl, and Sanda Harabagiu. 2006. The Impact of Question Decomposition on the Quality of Answer Summaries. In Proceedings of LREC 2006.

[13] Lao, Ni, Hideki Shima, Teruko Mitamura, and Eric Nyberg. 2008. Query Expansion and Machine Translation for Robust Cross-Lingual Information Retrieval, In Proceedings of NTCIR-7 Workshop.

[14] Mitamura, Teruko, Frank Lin, Hideki Shima, Mengqiu Wang, Jeongwoo Ko, Justin Betteridge, Matthew Bilotti, Andrew Schlaikjer and Eric Nyberg. 2007. JAVELIN III: Cross-Lingual Question Answering from Japanese and Chinese Documents. In Proceedings of NTICIR-6 Workshop.

[15] Mitamura, Teruko, Hideki Shima, Tetsuya Sakai, Noriko Kando, Tatsunori Mori, Koichi Takeda, Chin-Yew Lin, Ruihua Song, Chuan-Jie Lin, and Cheng-Wei Lee. 2010. Overview of the NTCIR-8 ACLIA Tasks: Advanced Cross-Lingual Information Access. In Proceedings of NTICIR-8 Workshop.

[16] Murdock, Vanessa and W. Bruce Croft. 2005. A Translation Model for Sentence Retrieval. In Proceedings of HLT-EMNLP 2005.

[17] Narayanan, Srini, and Sanda Harabagiu. 2004. Question Answering based on Semantic Structures. In Proceedings of COLING 2004.

[18] Pantel, Patrick, and Marco Pennacchiotti. 2006. Espresso: Leveraging Generic Patterns for Automatically Harvesting Semantic Relations. In Proceedings of COLING/ACL-06.

[19] Punyakanok, Vasin, Dan Roth, and Wen-Tau Yih. 2004. Mapping Dependencies Trees: An Application to Question Answering. In Proceedings of the 8th International Symposium on Artificial Intelligence and Mathematics, 2004.

[20] Ravichandran, Deepak. and Eduard. H. Hovy. 2002. Learning Surface Text Patterns for a Question Answering System. In Proceedings of ACL 2002.

[21] Sakai, Tetsuya, Hideki Shima, Noriko Kando, Ruihua Song, Chuan-Jie Lin, Teruko Mitamura, and Miho Sugimoto. 2010. Overview of NTCIR-8 ACLIA IR4QA. In Proceedings of NTCIR-8 Workshop.

[22] Schlaefer, Nico, Petra Gieselmann, Thomas Schaaf, and Alex Waibel. 2006. A Pattern Learning Approach to Question Answering within the Ephyra Framework. In Proceedings of the Ninth International Conference on TEXT, SPEECH and DIALOGUE (TSD) 2006.

[23] Schlaefer, Nico, Jeongwoo Ko, Justin Betteridge, Guido Sautter, Manas Pathak and Eric Nyberg. 2007. Semantic Extensions of the Ephyra QA System For TREC 2007. In Proceedings of the TREC 2007.

[24] Shen, Dan and Dietrich Klakow. 2006. Exploring Correlation of Dependency Relation Paths for Answer Extraction. In Proceedings of the ACL 2006.

[25] Shima, Hideki, Ni Lao, Eric Nyberg and Teruko Mitamura. 2008. Complex Cross-lingual Question Answering as Sequential Classification and Multi-Document Summarization Task, In Proceedings of NTCIR-7 Workshop.

[26] Sun, Renxu, Jing Jiang, Yee Fan Tan, Hang Cui, Tat-seng Chua, and Min-yen Kan. 2005. Using Syntactic and Semantic Relation Analysis in Question Answering. In Proceedings of the TREC 2005.

[27] Wang, Mengqiu, Noah A. Smith, and Teruko Mitamura. 2007. What is the Jeopardy Model? A Quasi-Synchronous Grammar for Question Answering. In Proceedings of EMNLP 2007.

[28] Zhang, Dell and Wee Sun Lee. 2002. Web Based Pattern Mining and Matching Approach to Question Answering. Proceedings of the Text REtrieval Conference 2002.