Learning to Extract Answers in Question Answering: Experimental Studies

Florent Jousse, Isabelle Tellier, Marc Tommasi and Patrick Marty[1]

Grappa Lab Lille 3 University and INRIA Futurs, Lille, Mostrare Project[2]

ABSTRACT. *Question Answering (QA) systems are complex programs able to answer a question in natural language. Their source of information is a given corpus or, as assumed here, the Web. To achieve their goal, these systems perform various subtasks among which the last one, called answer extraction, is very similar to an Information Extraction task. The main objective of this study it to adapt machine learning techniques defined for Information Extraction tasks to the slightly different task of answer extraction in QA systems. The specificities of QA systems are identified and exploited in this adaptation. Three algorithms, assuming an increasing abstraction of natural language texts, are tested and compared.*

RÉSUMÉ. *Les systèmes Question/Réponse sont des programmes complexes capables de répondre à une question en langage naturel, en utilisant comme source d'information soit un corpus donné, soit, comme c'est le cas ici, le Web. Pour cela, ces systèmes réalisent différentes sous-tâches parmi lesquelles la dernière, appelée extraction de la réponse, est très similaire à une tâche d'Extraction d'Information. L'objectif de cet article est d'adapter les techniques d'apprentissage automatique utilisées en Extraction d'Information à l'extraction de la réponse. Les spécificités des systèmes Question/Réponse sont identifiées et utilisées dans cette adaptation. Trois algorithmes utilisant une abstraction croissante du texte sont testés et comparés.*

KEYWORDS: *Question Answering, Machine Learning, Information Extraction*

MOTS-CLÉS : *Systèmes Question-Réponse, Extraction d'information, Apprentissage automatique*

1. This research was partially supported by: "CPER 2000-2006, Contrat de Plan état - région Nord/Pas-de-Calais: axe TACT, projet TIC"; fonds européens FEDER "TIC - Fouille Intelligente de données - Traitement Intelligent des Connaissance" OBJ 2-phasing out - 2001/3 - 4.1 - n 3.
2. www.grappa.univ-lille3.fr

1. Introduction

Question answering (*QA*) systems are complex systems that, given a question asked in natural language, can find an answer to this question, in a corpus or in the Web, and justify it by quoting their source(s). From the user's point of view, they can be considered as an improvement over traditional search engines such as Google or AltaVista because they provide a more direct and precise access to the desired information. The counterpart is that finding the correct answer to a question requires much more analysis and processing than a typical search engine.

Traditionally, a Question Answering system is divided into three steps: question analysis, passage retrieval and answer extraction. Our focus will be on the last task (answer extraction) which can be compared with an Information Extraction (*IE*) task. The purpose of *IE* is to automatically fill a database from a corpus of texts in natural language or semi-structured data from the Web. *IE* is an active research area in which many improvements have been made recently. Among them, we are particularly interested in machine learning techniques and techniques that deal with internet data sources [SOD 99, KUS 02, CAR 04].

Our objective is to improve the answer extraction task in *QA* systems on the Web by using insights provided by recent machine learning techniques developed for *IE*. Some specificities of *QA* systems can be useful here. As a matter of fact, in *QA* systems, the first steps of the process give hints that can help the answer extraction. Unlike in *IE*, we can take advantage of the outcomes of these steps. We focus our study on using common outcomes of the question analysis step: the class of the question and the keyword(s). To summarize, in this paper, we experimentally study how to use machine learning techniques developed for *IE* in answer extraction, taking into account *QA* systems specificities. We propose several ways to adapt existing *IE* algorithms, mainly differing by the document representation they adopt.

We have built a dataset to evaluate and analyze several alternatives. This dataset if freely available on `http://www.grappa.univ-lille3.fr/~jousse`. Our experiments deal with various encodings and three machine learning algorithms. The first algorithm, suggested by [RAV 02], applies on raw text. It relies on a document representation that is a sequence of token values (a token is either a word or a punctuation symbol) and builds extraction patterns by searching for the largest common subsequences of tokens present around the answer. *RAPIER*, defined by [CAL 98], is the second tested algorithm. It exploits deeper information about the documents, including part of speech tags, and elaborates complex extraction rules using both token values and *POS* tags. The third one is *PAF* developed in the Mostrare group [MAR 04]. *PAF* relies on supervised classification and is very flexible from the document representation point of view. In our experiments with *PAF*, token values are not used but only token types (*POS* tag, punctuation, case,...) and simple numerical features computable from the texts (lengths, *etc.*). These three algorithms are worth being compared because they make very different assumptions on how to represent a text in natural language. The representations used correspond to three levels of abstraction. For each algorithm, we also study the influence of using the keyword information.

The next section in an overview of *QA* systems. We present our experiments and our analysis in Section 3.

2. Overview of Question Answering Systems

Although the first works in this research field go back to the 1960s and 1970s with the works of [GRE 61], [WOO 73] and [LEH 78], Question Answering systems became popular in the late 1990s. This emergence was encouraged by the TREC-8 conference in 1999, where a *QA* track [VOO 99] was initiated and where the first large-scale evaluation of such systems took place. This evaluation led to a consensus on the general architecture of a *QA* system. Now, every *QA* system performs three successive tasks: question analysis, information retrieval and answer extraction. Although we will mainly focus on the third one, we introduce each of them and the assumptions made at each step, to justify our own choices. We also briefly review previous attempts to introduce machine learning techniques in *QA* systems.

2.1. *Question Analysis*

First, an analysis of the question is performed, its goal being to extract the information needed to perform the next two steps from the question. In all systems, the result is at least a classification of the question into a class. The set of possible classes is predefined, and ranges from few basic sets only depending on the first word of the question ("where", "when", etc.) to very fine sets of hierarchical classes.

But this information alone is not precise enough to directly allow to answer the question. So, another goal of this task is usually to extract words giving crucial information about the subject of the question. The notion of "focus" is often employed. We prefer to use the term *keywords*. A keyword is a word or a sequence of words. The number of keywords to be extracted from a question depends on its class. For example, in the question "When was W.A. Mozart born?", the question class can be `BIRTHYEAR` and the only keyword needed is "W.A. Mozart". So, the class `BIRTHYEAR` is of arity 1, meaning that only one keyword must be given with the class to specify a unique answer. This very precise class `BIRTHYEAR` could easily be replaced by a less precise one, for example `YEAR`, but with two keywords: "W.A. Mozart" and "born".

In the following, the assumption is made that the couple (question class, keywords) is *a synthetic representation of the question* and that this representation is necessary and sufficient to correctly answer the question. Some systems keep more precise a track of the initial question, such as AskMSR [BRI 02], including syntactic features. We do not pretend that a syntactic analysis is useless, for example to distinguish "Who killed Lee Harvey Oswald?" from "Who did Lee Harvey Oswald kill?". But the difference between both sentences can be reflected in different class assignments. The first question's representation could be (`NAME-KILLER`, Lee Harvey Oswald) while the second one's could be (`NAME-KILL`, Lee Harvey Oswald). These couples (question class, keywords) are the essential *QA*-specific data that will be used to perform the next steps.

2.2. *Information Retrieval*

Then, *QA* systems perform an *IR* task whose objective is to build a limited corpus of relevant documents, *i.e.* portions of texts in which the answer is very likely to be found. The purpose of this step is to *reduce the search space* of the next step. Selected passages must be long enough to contain the answer and some useful context. But when using the Web as a corpus, which is the case here, it is impossible to apply *NLP* techniques on the corpus beforehand. Therefore, the selected passages should not be too long, to be able to perform various *NLP* techniques such as *POS* tagging without dramatically increasing processing time.

2.3. *Answer Extraction*

Finally, the last task to be performed is the extraction of all candidate answers from the set of extracted passages. Due to the difficulty of the whole *QA* task, the candidate answers are then ranked to get the top 5 answers. This task is very similar to the one of Information Extraction, which consists in filling a database (whose structure is known) from natural language texts. In answer extraction, the answer can be considered as the (unique) piece of data to be extracted, and the question class represents its type.

Document representation plays a critical role in Information Extraction, hence also in answer extraction. Texts are usually considered as sequences of tokens, a token being either a word or a punctuation symbol. *POS* tags are often added to each token, and sometimes even syntactic structures are taken into account [LIG 04]. External resources like named entity recognizers or semantic information (such as Wordnet) can also be exploited [POI 03]. In our study, only *POS* tags and simple numerical descriptions computable from the texts (length, number of occurrences, *etc.*) will be used, in order to have a reasonable computation time.

An important question to consider is the representation of the keyword itself inside the passage. It is intuitively efficient to *abstract* it, that is to forget its value and replace it by either a type or a description that indicates "a part of text that contains the keyword". For example, in the question "When was Mozart born?", the question keyword is "Mozart". Abstracting it by a tag "⟨`KEYWORD`⟩" in documents allows to generate extraction rules like "*⟨KEYWORD⟩ was born in ⟨ANSWER⟩*" rather than "*Mozart was born in ⟨ANSWER⟩*". This seems much better because it is more generic, as it is correct for the other questions of the same class. One of our purposes is to test the efficiency of this abstraction.

The extraction itself is usually performed by hand-made patterns which describe the possible contexts in which the answer can occur. A set of patterns is associated with every question class. And, of course, the patterns themselves often include the abstracted keyword.

But the extraction cannot always be described in terms of patterns. One of the techniques we use is based on decision trees. Similarly, some question answering systems, such as Javelin [NYB 03], choose among different approaches (SVM, statistical extraction, patterns, *etc.*) depending on the question class in order to maximize their system's performances.

Finally, to rank the candidate answers, a confidence score is calculated. The score of a candidate answer can rely on its frequency, on its type (using a named entity tagger), as in [ABN 00]. This score can also rely on the proximity of the keyword(s), but this won't be the case in the following.

2.4. *Machine Learning and QA systems*

It is only recently that several attempts to introduce machine learning techniques to help process some of the tasks of a *QA* system have appeared. The advantages of introducing such techniques are obvious. The building of hand-made patterns used in question analysis and answer extraction tasks is long and fastidious. Learning them automatically would certainly be easier. Furthermore, employing learning techniques is the best way to automatically adapt a *QA* system to some specificities, such as: the language used (hand-made patterns are language-specific) and the domain of speciality (some patterns may also be specific to a domain) for specialized corpora.

Most of previous works trying to combine machine learning and *QA* focused on the first steps, for example on learning to automatically classify a question, or learning to locate and expand the keywords in the question in order to generate queries [USU 04]. To our knowledge, the first attempt to learn answer extraction patterns was the one of [RAV 02], using alignment learning with suffix trees.

But, we have seen that the answer extraction task shares many features with the *IE* task. In this domain, several machine learning techniques have been proposed. Our purpose is to adapt some of them to answer extraction, and to analyze their efficiency in this context.

3. Experiments

The main objective of this section is to use *IE* learning algorithms for answer extraction, and to evaluate the interest of using the keyword in those algorithms to make them more *QA* oriented. We will use three radically different *IE* learning algorithms. To evaluate the performances of the extraction rules learnt for each question class by these algorithms, we build a data set from documents provided by Google. We adapt the protocol proposed by [RAV 02] by making clearer the evaluation protocol, introducing *QA* cross-checking, in the spirit of standard machine learning cross validation.

3.1. *Building the corpus*

As we did not develop a complete *QA* system, the question analysis step is not implemented but only simulated. The experiments we make are based on only two predefined classes: `BIRTHYEAR` and `LOCATION`, both of arity 1. The inputs of our programs are thus couples (question class, keyword), which are supposed to correctly represent the initial questions. `BIRTHYEAR` is supposedly an easy question class because of its very specific aspect, whereas `LOCATION` is considered more difficult, since it includes localisation of towns, sites, lakes, *etc.*

For each class, we choose 100 distinct keywords. Each keyword is associated with a list of all its valid answers, called $Answers$. For example, the question (`LOCATION` , "Danube") has several valid answers, such as "Hungary", "Austria", *etc.*

Algorithm 1 Building the Learning Corpus for a Given Question Class

```
Input : the set $S$ of keyword/Answers couples.
1:  $Corpus = \emptyset$
2:  for each couple $(keyword, Answers)$ in $S$ do
3:      $Documents$ = top 100 documents retrieved by Google containing both
        $keyword$ and at least one $answer \in Answers$
4:      for each $document$ in $Documents$ do
5:          for each $answer$ in $Answers$ do
6:              Let $s$ be the smallest passage containing both $keyword$ and $answer$
7:              Let $s_2$ be $s$ with surrounding context (50 characters) from $document$
8:              tokenize($s_2$)
9:              Add $s_2$ to $Corpus$
10:         end for
11:     end for
12: end for
Output : $Corpus$
```

To build the learning corpus, we adapt the method proposed in [RAV 02]. We first build a corpus of HTML documents containing both the keyword and at least one of the valid answers. These are obtained by sending Google a query consisting of the keyword and a disjunction of all the valid answers. For example, with the keyword "Great Pyramid" and the answers "Giza" and "Egypt", the query sent to Google is " 'Great Pyramid' 'Giza' OR 'Egypt' ". Query expansion techniques, based on both the keyword(s) and the class of the question, are often used at this step to increase the recall, but we do not use any of them here.

Then, inside these HTML documents, all HTML tags are removed in order to only keep natural language texts. The smallest passages containing both the keyword and a valid answer are selected, leaving some context (50 characters) before and after this passage. For example, if the text is "the great composer Ludwig Von Beethoven was born in 1770 in Bonn, Germany.", the selected passage is something like "*great composer* Ludwig Von Beethoven was born in 1770 *in Bonn*" (the context is in italics).

The next step consists in tokenizing these passages to match our document representation. A document is considered as a sequence of tokens. As already stated, a token is either a word or a punctuation symbol. Spaces, tabulations, line feed, *etc.* are not considered as tokens but as token delimiters and hence are not taken into account.

The opportunity to realize abstraction of the keyword as presented in Section 2.3 leads us to build two versions of each corpus. Finally, document preparation as well as the way to use the keyword differ with the choice of the learning algorithm.

3.2. *Evaluation protocol: QA cross-checking*

In order to evaluate the performance of the rules learnt by each tested algorithm, with or without using the keyword, we define a new evaluation protocol, inspired by machine learning cross-validation, and independent of the learning algorithm.

Because of the way we built the learning corpus, it is mandatory to build a new separate testing corpus. Indeed, if we separated the corpus built as described in the previous section into two distinct parts, a learning corpus and a testing corpus, a typical cross validation technique could be applied. But this corpus is exclusively made of passages containing both the keyword and a valid answer. So, the results obtained would necessarily differ from the ones obtained in a real QA process, where the passages extracted in the information retrieval step are selected without knowing the answer, and hence contain the keyword but not necessarily a valid answer. To avoid this cheat, a new testing corpus must be built.

Building the testing corpus is quite similar to building the learning corpus. We only change two aspects of algorithm 1. First, for each question, we send Google a query consisting of the keyword alone. Second, the keyword is used to retrieve small passages. In each document, the selected passages are fixed-sized windows around all the occurrences of the keyword. The window size is clearly a parameter to adjust. Roughly, larger is the window, better is the recall but worst is the precision. We have made experiments with several window sizes that confirm this intuition. The results presented here are for a window size of 200 characters before and after the keyword.

Finally, the complete evaluation protocol is given in algorithm 2. We have two corpora, the learning corpus L and the testing corpus T, built from a set S of 100 (keyword,Answers) couples c_0 to c_{99}. In both corpora, each passage is associated to a couple. Hence, each corpus is divided into 10 parts: l_0 to l_9 for the learning corpus, and t_0 to t_9 for the testing corpus, each part i containing passages associated to 10 couples c_{10i} to c_{10i+9}. This way we build a partition of our corpora. Then 10 iterations are performed. In each iteration k, extraction rules are learnt on $L \setminus l_k$, and

Algorithm 2 *QA* Cross-Checking for a Given Question Class

```
Input : $S = \{c_j | j \in [0, 99]\}$, the set of keyword/Answers couples.
        $L$, the learning corpus.
        $T$, the testing corpus.
 1: Build the partition of $L = \{l_i | i \in [0, 9]\}$, where $l_i$ is the set of passages of $L$
    corresponding to the keyword/Answers couples $c_{10i}$ to $c_{10i+9}$
 2: Build the partition of $T = \{t_i | i \in [0, 9]\}$, where $t_i$ is the set of passages of $T$
    corresponding to the keyword/Answers couples $c_{10i}$ to $c_{10i+9}$
 3: for $i$ from 0 to 9 do
 4:     $Rules$ = Learn($L \setminus l_i$).
 5:     for each $c_j$ represented in $t_i$ do
 6:         Apply $Rules$ on the passages of $t_i$ corresponding to $c_j$.
 7:         Rank the extracted candidate answers.
 8:         $RR_j$ = Reciprocal Rank of the correct answer.
 9:     end for
10: end for
Output : $MRR$ = Mean of the $RR_j$
```

tested on t_k. This way, each passage of the testing corpus is tested only once and rules are never evaluated on the questions they have been learnt with.

Finally, to evaluate the performances of the extraction rules, we use a metric called Mean Reciprocal Rank (MRR), which is the one used in the TREC *QA* track [VOO 99]. For each question, the score is the Reciprocal Rank (RR) of the correct answer, meaning that if the correct answer ranks 2^{nd}, the score is $\frac{1}{2}$. The MRR is the mean of these scores. To evaluate whether an extracted answer is correct, we perform an "exact match" evaluation, *i.e.* an extracted answer is considered correct if it is identical to the correct answer. For example, if the expected answer is "Paris", "paris" will be considered as correct, but "in Paris" will not. Moreover, for each question, only the best ranked correct answer counts. For example, if the expected correct answers are "Paris" or "France", and the system has extracted "Paris", "Lille" and "France", the correct extracted answer "France" won't count since "Paris" has also been extracted and ranked higher.

3.2.1. *Statistics*

Table 1 shows statistics on our learning and testing corpora. Every passage of the learning corpus contains both the keyword and at least one correct answer, whereas every passage of the testing corpus contains the keyword and hopefully the answer, but it is not compulsory. In each iteration of our evaluation with `BIRTHYEAR` questions, we learn on approximately 1900 annotated passages and test on appromimately 2900 passages, while with `LOCATION` questions, we learn on 3000 annotated passages and test on 2000 passages.

We can also notice that only 93 LOCATION questions have at least one corresponding passage in the testing corpus containing the correct answer. Therefore, 7 questions will not be answered in the answer extraction step.

		BIRTHYEAR	LOCATION
Learning Corpus	# passages	2096	3374
Testing Corpus	# passages	29068	20727
	# passages with answer	1770	5546
	# Questions having at least one passage with a correct answer	100/100	93/100

Table 1. *Corpora Statistics*

Now that we have described the building of our corpora and our evaluation protocol, we will start our experiments with three different Information Extraction algorithms.

3.3. *Experiments with Alignment Learning using Suffix Trees*

3.3.1. *Learning Algorithm*

The first algorithm exploits a simple representation of documents as a sequence of token values. The underlying principles of the algorithm are also easy. It consists in building rules based on frequent subsequences around the answer. This approach is used in [RAV 02]. The answer is replaced by the tag ⟨ANSWER⟩ in order to be able to locate regularities around it. Building the suffix tree of all the passages in the learning corpus allows to find these frequent subsequences in linear time. Only the subsequences of tokens containing the tag ⟨ANSWER⟩, occuring in at least 5 passages, and containing at least 3 tokens (to avoid learning over general patterns, such as the pattern "⟨ANSWER⟩"), are kept. These subsequences form a set of patterns, which must then be ranked to build extraction rules. To do so, we measure the patterns precision by applying them one by one to the passages we learnt them with. The precision of a pattern is the ratio between the number of correct extractions and the total number of extractions. Patterns are ranked from the more precise to the least precise. The pseudo-code of this algorithm is given in Fig 3.

Algorithm 3 Alignment Learning Algorithm

```
Input : List of passages.
1: Search for subsequences occurring in multiple passages.
2: $Patterns$ are the subsequences containing the answer and at least $3$ tokens long.
3: Compute the confidence score of the $Patterns$, which is its precision.
4: Sort the $Patterns$ according to their confidence score.
Output : Ordered list of patterns along with their confidence score.
```

To take advantage of the keyword in this algorithm, we use the method introduced by [RAV 02], which differs from the algorithm in Fig 3 on two points. First, the keyword is abstracted. It is replaced by the tag ⟨`KEYWORD`⟩ in the corpus and only frequent subsequences containing both the tags ⟨`KEYWORD`⟩ and ⟨`ANSWER`⟩ are kept. The second difference concerns the pattern ranking. Instead of measuring the precision on the learning corpus, we measure it on the corresponding part of the testing corpus. This means the precision is measured on passages associated with the same questions as the learning corpus, but containing the keyword and not necessarily the answer (*c.f.* 3.1). These two changes make the algorithm more adapted to a QA system.

3.3.2. *Extraction algorithm*

Both with and without using the keyword, once the rules have been applied on the documents, we have a list of all the occurrences of the candidate answers along with their confidence score, which is the precision of the pattern which extracted it. To rank these candidate answers, we compute a score for each candidate answer which is the sum of the confidence scores of all the occurrences of this candidate answer, as shown in Fig 4. For example, if the candidate answer "1756" has been extracted twice with the score 0.5 and 0.3, the score of "1756" is 0.8.

Algorithm 4 Ranking Algorithm

```
Input : List of answers along with their confidence score.
1: for each $(answer, confidence)$ do
2:     add $confidence$ to $score_{answer}$
3: end for
4: Sort the answers according to $score_{answer}$
Output : Ordered list of the candidate answers.
```

3.3.3. *DataSet Pre-processing*

In each passage of our learning corpus, the answer was replaced by the tag ⟨`ANSWER`⟩ for both experiments with and without using the keyword information. When using it, we also replaced it by the tag ⟨`KEYWORD`⟩ in both our learning and testing corpora.

3.3.4. *Results*

Table 2 shows the results of our experiments. With both question classes, our results when using the keyword are not as good as those in [RAV 02] but the experimental conditions (corpora, protocols) are too different to allow a reliable comparison.

One can notice that with both question classes, the rules learnt using the keyword perform better. With `LOCATION` questions, the MRR is almost three times better, and the system correctly answers 23 more questions than when the keyword is not used. With `BIRTHYEAR` questions, the difference is less significative. Indeed, the MRR is very close but yet 8 more questions are answered. Looking closer at the extracted answers shows that with the patterns learnt with the keyword information, 21 answers

Keyword	BIRTHYEAR		LOCATION	
	#Answers	MRR	#Answers	MRR
not used	55/100	0.452	34/93	0.172
used	63/100	0.485	57/93	0.513

Table 2. *Results using Alignment Learning with Suffi x Trees. Column #Answers gives the number of times at least one of the correct answers belongs to the top 5 answers*

are considered incorrect because they contain the exact birth date and not just the birth year, for example "January 20 , 1946" instead of "1946", against 5 with the rules learnt without using the keyword. Considering these answers correct brings the MRR to 0.672 with the keyword information and 0.475 without.

Moreover, with BIRTHYEAR questions and the rules learnt with the keyword information, we notice that, in most cases, when a correct answer has been extracted but is not in the top 5 answers, which happens 13 times, all the answers ranked higher are just noise and none of them are dates, not even numbers. This particular case shows the main weakness of these patterns. Indeed, they are based on raw text only and do not provide any information concerning the answer to be extracted. Adding information about the answer would help solve this problem. For example, with BIRTHYEAR questions, the *POS* tag could tell that the answer must be a number.

Overall, looking at the patterns makes it easy to explain why those learnt using the keyword perform better on both question classes. Indeed, both on BIRTHYEAR and on LOCATION questions, when the keyword is not used, the algorithm learns patterns that are very precise but overly specific. Most of the patterns contain the value of the keyword, for example "Adams (⟨ANSWER⟩" or "Hitchcock (⟨ANSWER⟩" with BIRTHYEAR questions, and "Parthenon , ⟨ANSWER⟩" or "Balaton , ⟨ANSWER⟩" with LOCATION questions. Since the keyword is not abstracted, these patterns are too specific and cannot be used to answer different questions. When the keyword is abstracted, the patterns are more general and the previous patterns are replaced by "⟨KEYWORD⟩ (⟨ANSWER⟩" for BIRTHYEAR questions and "⟨KEYWORD⟩ , ⟨ANSWER⟩" for LOCATION questions.

3.4. *Experiments with RAPIER*

3.4.1. *Algorithm*

The second approach consists in using the *IE* system *RAPIER* [CAL 98] to learn answer extraction patterns and extract the answers. *RAPIER*'s patterns have a more abstract view of texts. Indeed, not only do they use the value of the token, but they also use its *POS* tag. These patterns are sequences of tokens on which there are two types of constraint, each constraint being a disjunction of possible values. The first constraints are *word constraints* and concern the value of the token. For example, if the constraint is a list of the words a, b and c, then the token must be one of these three words. The second constraints are called *Part-Of-Speech constraints* and indicate which *POS* tag the token must have. Having these two constraints allows the system to generalize his patterns and, for example, learn a pattern saying that a token must have the *POS* tag "NN" (noun) without explicitly giving its value. The reader is reported to [CAL 98] for further explanation on how *RAPIER* works.

Since *RAPIER* does not give a confidence score for each candidate answers, we cannot rank the candidate answers as we did with alignment learning. But *RAPIER*'s patterns set is ordered, so we rank our candidate answers accordingly to the rank of the pattern that extracted them. For example, if "1756" is extracted by rule 1 and "1845" by rule 3, the best candidate answer is "1756".

3.4.2. *DataSet Pre-processing*

To meet *RAPIER* requirements, we had to add *POS* tags to our passages. We used Brill's *POS* tagger [BRI 92] to do so. To take advantage of the keyword information, we also abstracted it by replacing all its occurrences by the tag ⟨`KEYWORD`⟩.

3.4.3. *Results*

	`BIRTHYEAR`		`LOCATION`	
Keyword	#Answers	MRR	#Answers	MRR
not used	66/100	0.491	10/93	0.099
used	74/100	0.579	11/93	0.085

Table 3. *Results using RAPIER. Column #Answers gives the number of times at least one of the correct answers belongs to the top 5 answers*

Table 3 shows the results. The first thing to notice is that *RAPIER* does not perform well on `LOCATION` questions. Indeed, whether the keyword information is used or not, the results are equally poor, meaning it is totally independent of the keyword. The problem in this case is that *RAPIER* never generalizes on the answer to extract, *i.e.* in all the patterns it learns, the word constraints on the tokens to be extracted are lists of the answers encountered during the learning process. Therefore, the patterns are far too specific to the questions in the learning corpus, and hence cannot be used to answer other questions. Here, *RAPIER* may need more examples to learn on in order to perform a proper generalization and learn more efficient patterns.

Oppositely, on `BIRTHYEAR` questions, the system performs well and using the keyword improves the performances. For example, whether the keyword is used or not, *RAPIER* is able to learn a pattern saying that the answer is a number, preceded by a "(" and followed by a "-", as in "Mozart (1756 - 1791". But when using the keyword, *RAPIER* is also able to learn that the token preceding the "(" is precisely the keyword, whereas when the keyword is not abstracted, *RAPIER* is at best able to learn that this token is a proper noun. Unfortunately, this is the only generalized pattern produced by *RAPIER* in our experiments. All the other patterns are over specific for the same reason as with `LOCATION` questions. Once again, a larger set of examples may have helped *RAPIER* to learn more general patterns. But providing larger sets of examples is difficult for some question classes and may be expensive.

3.5. *Experiments with PAF*

3.5.1. *Algorithm*

The third learning algorithm we used is *PAF* [MAR 04]. This algorithm's approach is very different from the previous two algorithms. Indeed, it does not learn patterns, but classifiers. The passages of our corpora being tokenized, a separator is defined as the position between two successive tokens. Thus, to identify the correct answer, we need to identify its start separator and its end separator. Separators are represented as an attribute-valued vector. *PAF* learns to identify separators using supervised classification. The classifiers learnt are readable decision trees produced by Quinlan's C5. To rank the candidate answers given by *PAF*, a score for each candidate answer is computed the same way as was done with alignment learning.

3.5.2. *DataSet Pre-processing*

For the classifier to be able to effectively learn how to classify the separators, we had to choose a document representation, *i.e.* a list of attributes. In our experiments, we considered the following attributes for each token:

– the Part-Of-Speech tag (Brill's *POS* tagger [BRI 92] was used);

– an attribute saying whether the token is a word, a number or a punctuation symbol. This is a generalization over the Part-Of-Speech tag;

– the case of the token: allCaps, lowercase, UpperInitial or LowerInitial;

– the length of the token (i.e. its number of characters).

Note that this document representation is an abstraction of the initial text, as it does not use the string value of the tokens. Hence, unlike *RAPIER*, *PAF* cannot learn lists of the answers encountered in the learning corpus.

To use the keyword, we could not abstract it since the string values of the tokens are not considered. Therefore, we added an attribute that represents the distance between the keyword and the answer. For example in the passage "Mozart (1756 -" the distance is 1 token. The order in which the keyword and the answer appear is also represented. In our example, the keyword is before the answer.

3.5.3. *Results*

Table 4 shows the results, which outperform the other ones. On the one hand, with `BIRTHYEAR` questions, the system performs very well and the keyword attribute helps improve the results. At the root of the decision tree learnt without using the keyword attribute, one can read that the answer must be a number of length 4, which is not

	BIRTHYEAR		LOCATION	
Keyword	#Answers	MRR	#Answers	MRR
without keyword attribute	75/100	0.557	71/93	0.513
with keyword attribute	82/100	0.601	68/93	0.506

Table 4. *Results using PAF. Column #Answers gives the number of times at least one of the correct answers belongs to the top 5 answers*

preceded by another number. The second condition is used to eliminate the death year, which often follows the birth year. When using the keyword attribute, we have more precise answers since *PAF* also learns, for example, that when a 4-digit number is preceded by the keyword and a "(", it is the correct birth year.

On the other hand, results with `LOCATION` questions are more surprising. Indeed, the system performs slightly better without using the keyword attribute. But first, we also notice that the MRR is almost the same in both cases. Moreover, whether the keyword attribute is used or not, the system extracts the same answers, the only difference being that, when using the keyword attribute, three of them are not ranked in the top 5. So the difference may lie more in the ranking than in the extraction itself. Finally, this result does not mean, of course, that the keyword is useless. Remember that all the passages of the testing corpus consist in windows around the keyword. So the keyword has in fact already been used, and in this particular case, passage selection might be precise enough to help *PAF* locate the correct answer.

4. Conclusion and Future Works

Our experiments show that in every case, with these three algorithms, using the keyword helps them learn better extraction rules. Indeed, knowing the keyword allows the rules to be less specific to the questions in the learning corpus, and hence more efficient when used to answer questions that are not in the learning corpus. However, this improvement seems to be less significant when using a rich document representation, as with *PAF*.

Somehow, our experiments indicate that rather than tuning or changing the learning algorithm in the extraction step, the most crucial choice should be the adaptation of the document representation given the question class. *PAF* approach is well-suited to do so. The results with it tend to show that on the corpus we built, a document representation which abstracts the initial text but which is rich enough (*POS* tags, punctuation, *etc.*) might give better results and flexibility.

Overall, we need to do further experiments to confirm these results. First, experimenting with other question classes will help validate that our results are independent of the question class. Moreover, we could enrich our document representation by tagging named entities, to validate that a richer representation gives better results.

5. References

[ABN 00] ABNEY S., COLLINS M., SINGHAL A., "Answer extraction", *ANLP-2000*, 2000.

[BRI 92] BRILL E., "A Simple Rule-Based Part Of Speech Tagger", *Proceedings of the Third Conference on Applied Computational Linguistics (ACL)*, 1992.

[BRI 02] BRILL E., DUMAIS S., BANKO M., "An analysis of the AskMSR question-answering system", 2002.

[CAL 98] CALIFF M. E., "Relational Learning Techniques for Natural Language Information Extraction", report num. AI98-276, August 1998, Artifi cial Intelligence Laboratory, University of Texas of Austin.

[CAR 04] CARME J., LEMAY A., NIEHREN J., "Learning Node Selecting Tree Transducer from Completely Annotated Examples", *7th International Colloquium on Grammatical Inference*, Lecture Notes in Artifi cial Intelligence, Springer Verlag, 2004.

[GRE 61] GREEN B. F., WOLF A. K., CHOMSKY C., LAUGHERY K., "BASEBALL: An Automatic Question Answerer", *Proceedings of the Western Joint Computer Conference 19*, 1961, p. 219-224.

[KUS 02] KUSHMERICK N., "Finite-state approaches to Web information extraction", *Proc. 3rd Summer Convention on Information Extraction*, 2002.

[LEH 78] LEHNERT W., *The Process of Question Answering: A Computer Simulation of Cognition*, Lawrence Erlbaum Associates, 1978.

[LIG 04] LIGOZAT A.-L., "Système de Question-Réponse : Apport de l'Analyse Syntaxique à l'Extraction de la Réponse", *conférence RECITAL 2004*, Fes, avril 2004.

[MAR 04] MARTY P., TORRE F., "Codages et connaissances en extraction d'information", *Actes de la Sixième Conférence Apprentissage CAp'2004*, 2004, p. 207-222.

[NYB 03] NYBERG E., MITAMURA T., CALLAN J., CARBONELL J., FREDERKING R., COLLINS-THOMPSON K., HIYAKUMOTO L., HUANG Y., HUTTENHOWER C., JUDY S., KO J., KUPSC A., LITA L. V., PEDRO V., SVOBODA D., , DURME B. V., "The JAVELIN Question-Answering System at TREC 2003: A Multi-Strategy Approach with Dynamic Planning", *Proceedings of TREC 12*, November 2003.

[POI 03] POIBEAU T., *Extraction Automatique d'Information*, Hermes, Paris, 2003.

[RAV 02] RAVICHANDRAN D., HOVY E., "Learning Surface Text Patterns for a Question Answering System", *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2002.

[SOD 99] SODERLAND S., "Learning Information Extraction Rules for Semi-Structured and Free Text", *Machine Learning*, vol. 34, num. 1-3, 1999, p. 233-272.

[USU 04] USUNIER N., AMINI M., GALLINARI P., GRAU B., "Génération de requêtes pour les systèmes de Q/R avec un modèle d'apprentissage statistique", *Workshop Question-Réponse TALN 2004*, Fez, Maroc, 2004.

[VOO 99] VOORHEES E., TICE D., "The TREC-8 question answering track evaluation", 1999.

[WOO 73] WOODS W., "Progress in Natural Language Understanding - An Application to Lunar Geology", *AFIPS Conference Proceedings, volume 42*, 1973, p. 441-450.