Question Classification by Structure Induction[*]

Menno van Zaanen and **Luiz Augusto Pizzato** and **Diego Mollá**
Division of Information and Communication Sciences (ICS)
Department of Computing
Macquarie University
2109 North Ryde, NSW
Australia
{menno, pizzato, diego}@ics.mq.edu.au

Abstract

Most current question answering systems analyze questions to understand what kind of answer the user would like to receive. Once the expected type of the answer belonging to a question is known, the number of possible answers that need to be considered can be reduced.

In this article we will introduce a new approach (and several implementations) to the task of question classification. The approach concentrates on the structural information that is present in the questions. This information is extracted using machine learning techniques and the patterns found are used to classify the questions. The approach fits in between the machine learning and handcrafting of regular expressions (as it was done in the past). It combines the best of both: classifiers can be generated automatically and the output can be investigated and manually optimized if needed. This research is performed within the AnswerFinder project, which concentrates on symbolic approaches to question answering.

1 Introduction

Question Answering (QA) systems try to find answers to natural language questions by searching natural language texts. With the availability of big amounts of data, this becomes useful from a user perspective and interesting from a research perspective. On the one hand, QA systems make life easier for a user in that the system actually finds *answers* instead of relevant documents as with Information Retrieval (IR). The implementation of QA systems, on the other hand, requires the combination of many NLP tools (which may include IR techniques).

Many QA systems are implemented based on a generic architecture. Initially, the documents are prepared off-line, extracting the essence of the contents and allowing easier retrieval. When the user presents a question to the system, the question is analyzed first. This results in a question image that is used to pre-select a number of relevant documents from the document pool. From these selected documents, the possible answers are extracted. This is done, among others, on the basis of the expected answer type (EAT), such as "location", "time", "number", which is also found during question analysis. Once the possible answers are extracted, they are scored and re-ordered. Finally, the best answer (or set of answers) is returned to the user.

Before a QA system can answer a question, it needs to have an idea what the question is about. Not only must the system find the focus of the question, it should also know the EAT. Finding the EAT of a question is called question classification (or EAT classification) [Hermjakob, 2001]. In the EAT classification step in the question analysis phase, the question is examined and one of the possible answer types is assigned to it. Classes can be, for example, "number", or "location", but they can also be more fine-grained: "number-distance", "number-weight", "location-city", or "location-mountain". The final answer of the system should belong in the same EAT class as the question.

Here, we will introduce a new approach to the problem of EAT classification, based on structural information that can be extracted from the questions. Re-occuring structures found in questions, such as "How far . . . " may help finding the correct EAT (in this case "distance") for a particular question. The approach described here automatically finds these structures during training and uses this information when classifying new questions.

In the next section, the main approach will be described, as well as two different systems (with corresponding implementations) based on this idea. Next, the implementations are applied to real data and the results of the systems are given. We will conclude with a discussion of possible future work and an overview of the advantages and disadvantages of these systems.

2 Approach

Past approaches to EAT classification can be roughly divided into two groups: *machine learning* and *regular expression* approaches. Grouping questions in a fixed set of answer types is typically a classification task for which "standard" machine learning techniques can be used. In fact, this has been done with very good results [Li and Roth, 2002; Zhang and Sun Lee, 2003]. Alternatively, handcrafted regular expressions that indicate when questions belong to a cer-

[*]This work is supported by the Australian Research Council, ARC Discovery grant no. DP0450750.

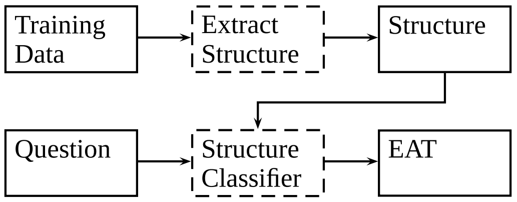

Figure 1: Overview of the structure induction approach

tain class can also be used [Voorhees and Buckland, 2002; TREC, 2003]. As an example, one might consider the regular expression `/^How far/` that can be used to classify questions as having an EAT of "distance".

The regular expression approach has the advantage that regular expressions are human readable and easy to understand. However, creating them is not always trivial. Having many fine-grained classes makes it extremely hard for regular expressions to distinguish between closely related classes.

The machine learning approach produces good results and classifiers are easy to create (when training data is available). Several standard techniques can be used directly [Mitchell, 1997]. Questions are converted into (fixed length) feature vectors that describe the important aspects of the question. The classifier takes this feature vector and assigns a class to it (i.e. the EAT). How the decision is made, depends on the technique that is applied.

Of course, there are also other, more complex, approaches to EAT classification, such as the one described in [Harabagiu *et al.*, 2000]. In that system, WordNet information from the words in the question is used to find the EAT. This is similar to the regular expressions approach, where words are matched in the question, but much additional linguistic information and world knowledge is added to the process.

In this article, we describe a new approach that is a combination of both approaches. Using machine learning, patterns are extracted from the training data. These patterns serve as regular expressions during the classification task. An overview of the approach is given in Figure 1.

In the next two sections, we will describe two systems that fit into this approach. The first one uses a grammatical inference system to find structure in the questions and the second one is based on tries. It finds regularities in the trie and classifies using these regularities.

2.1 Alignment-Based Learning classifier

The structure extraction phase of the first system is done by Alignment-Based Learning (ABL). ABL is a generic grammatical inference framework, that learns structure in the form of pairs of brackets using plain text only. It has been applied to several corpora in different languages with good results [van Zaanen, 2002] and it also compares well against other grammatical inference systems [van Zaanen and Adriaans, 2001].

The underlying idea of ABL is that constituents can be interchanged. To give an example, if we exchange the noun phrase *the man* in the sentence *He sees the man* with another noun phrase *a woman*, we get another valid sentence: *He sees a woman*. This process can be reversed (by aligning sentences) and possible constituents, called hypotheses can be found. This is called the alignment learning phase.

Table 1: Example sentences with ABL structure

"DESC"	*(What) (is (caffeine))*
"DESC"	*(What) (is (Teflon))*
"LOC"	*(Where) is (Milan)*
"LOC"	*What (are the twin cities)*

The ABL framework consists of two more phases: clustering and selection learning. The clustering phase groups similar non-terminal labels that are assigned to the hypotheses during the alignment-learning phase. The selection learning phase resolves a problem of the alignment learning phase. The alignment learning phase may introduce overlapping hypotheses. These are unwanted if the underlying grammar is considered context-free and the resulting structure is taken as a parse of the sentence using the underlying grammar. The selection learning phase attempts to remove the incorrect overlapping hypotheses. Both phases are not used in this article, we only concentrate on the alignment learning phase.

Several methods of the alignment learning phase have been implemented. Due to practical restrictions of time and space (memory) [van Zaanen, 2003], we have used the suffixtree implementation [Geertzen and van Zaanen, 2004]. Here, the alignments are found by building a suffixtree, which uncovers re-occurring sequences of words in the sentences. Using these sequences, the hypotheses are introduced.[1]

Once the structure has been found, it is extracted and used in classification. Each training question is associated with an EAT, so the structure extracted from a certain question provides some evidence that this structure is related to the EAT. If a structure occurs with several EATs (i.e. it occurs in multiple questions that have different EAT), the structure is stored with all EATs and their respective frequencies.

The stored structures can be seen as regular expressions, so during classification, all structures are matched on the new question. If a regular expression matches, its frequency information is remembered and when all structures have been tried, the class with the highest frequency is selected as output.

We will explain the different implementations by walking through an example. Consider the questions combined with the structure found by ABL and their EAT depicted in Table 1.

The first implementation, which we will call *hypo*, takes the words in the hypotheses (i.e. the words between the brackets), turns them into a regular expression and stores it together with the EAT of the question. The resulting regular expressions can be found in Table 2. For example, the first two sentences both generate the regular expression `/What/` combined with EAT "DESC", so it has frequency 2.

The second implementation, called *unhypo*, takes each hypothesis and removes it from the sentence when it is turned into a regular expression. The regular expressions extracted from the example sentences can also be found in Table 2. For

[1]For more information on ABL and the implementations of the different phases, see [van Zaanen, 2002; Geertzen and van Zaanen, 2004].

Table 2: Regular expressions found by ABL-based methods

hypo			unhypo		
caffeine	“DESC”	1	What is	“DESC”	2
is caffeine	“DESC”	1	What	“DESC”	2
What	“DESC”	2	is caffeine	“DESC”	1
Teflon	“DESC”	1	is Teflon	“DESC”	1
is Teflon	“DESC”	1	Where is	“LOC”	1
Milan	“LOC”	1	is Milan	“LOC”	1
Where	“LOC”	1	What	“LOC”	1
are the twin cities	“LOC”	1			

example, the first two questions would introduce `/What/` combined with EAT “DESC” and frequency 2. The last question would introduce `/What/` with class “LOC” and frequency 1.

Once the structures are stored with their class and frequency, the actual classification can start. We have built two methods that combine the evidence for EATs of the question differently. Both start with trying to match each of the stored regular expressions against the question. If a regular expression matches, the first method (called *default*) will increase the counts of the EATs of the question with the frequency that is stored with the EATs of the regular expression.

The second method (called *prior*) will increase the counts of the EATs of the question that are related to the regular expression with 1 (where the default method would increase it with the frequency count of each EAT). When all regular expressions are tried, the default method selects the EAT with the highest frequency (if there are more with the same frequency, one is chosen at random), whereas the prior method also selects the one with the highest count, but if there are more than one, it selects one based on the EAT with the highest overall frequency.

Part-of-speech

In addition to applying the systems on the plain text, we also apply them to tokens consisting of word combined with their part-of-speech (POS).

We create the POS information using Brill’s tagger [Brill, 1992]. The POS information is simply combined with the plain words. However, adjacent words that have the same POS are combined into one token. We will clarify this with an example. The question *Who is Federico Fellini?* is, after POS tagging, divided into three tokens: (*Who*, WP), (*is*, VBZ) and (*Federico Fellini*, NNP). *Federico* and *Fellini* are combined in one unique token, because *Federico* and *Fellini* are adjacent words that have the same POS.

2.2 Trie classifier

The other system we describe here is based on finding structure using a trie. By searching this data structure, it can also be used to classify new questions. The work discussed here is an extension of the work in [Pizzato, 2004].

A trie $T(S)$ is a data structure defined by a recursive rule $T(S) = \{T(S/a_1), T(S/a_2), \ldots, T(S/a_r)\}$. S is a set of sequences, whose elements are taken from the alphabet A. S/a_n is the set of sequences that contains all sequences of S

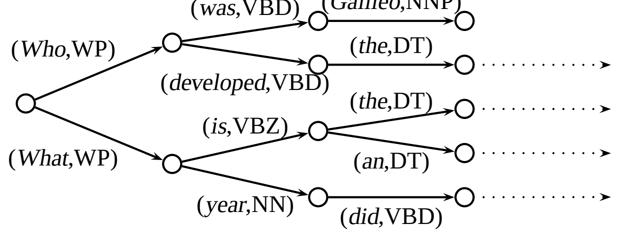

Figure 2: Fragment of the Question-Trie

that start with a_n, but stripped of that initial element [Clément *et al.*, 1998].

All the questions are combined in one trie structure. Figure 2 contains a fragment of such a trie structure. In addition to the token, the EAT and frequency information (the number of questions that use that path in the trie), taken from the training data, are also stored in the nodes in the trie. Each node in the (unique) path of a certain question holds this information.

The classification of new questions is performed by extracting the EAT information stored in the trie nodes. The tokens of the new question are used to find a path in the trie. The EAT information can then be extracted from the final node of the path through the trie. To be more concrete, consider a question $Q = (q_1, q_2, \ldots, q_n)$ and a trie $T(S)$. The classification of the question is done by finding the path $T(S/q_1/q_2/\ldots/q_n)$. The EAT information stored in the final node of the path is returned.

Notice that only with the questions present in the training data, we can find a complete path in the trie. To find a path for questions that have not been seen in the training data, we skip non-matching tokens wherever needed to complete the path. This is called the look-ahead process.

The look-ahead process works as follows. Let us say that the question tokens of question Q match up to q_n: $T(S/q_1/q_2/\ldots/q_{n-1}/q_n)$ and $T(S/q_1/q_2/\ldots/q_{n-1}/q_n/q_{n+1})$ does not exist. The look-ahead process then builds a set of sub-tries of the form $T(S/q_1/q_2/\ldots/q_{n-1}/q_n/\beta/q_{n+2}) \mid \beta \in A$.

One of these sub-tries is selected based on the frequency of the prefix defined by the sub-trie in the training data. In other words, for each sub-trie in the set, the prefix is extracted $(q_1, q_2, \ldots, q_{n-1}, q_n, \beta, q_{n+2})$ and the frequency of this prefix in the training data is looked up. The sub-trie that has the prefix with the highest frequency associated to it, is selected. This defines β and the process continues with the next question token until all tokens are consumed.

Here, we will describe two different implementations of the EAT classification system using a trie structure and POS information. The first uses a method, called *strict*, where a question token can only be replaced (if needed) by another token if they belong to the same POS class. That is to say, in the example above β and q_{n+1} must have the same POS tag. If there are more choices, the one that has the highest frequency is selected. If no option is found at all, the search process stops at that point and the EAT information is retrieved from node q_n in the trie.

The second trie-based implementation, called *flex*, allows a token in the question to be replaced with one or more tokens in the trie. Initially, it works similar to the strict method. However, if none of the possible β tokens has the same POS at the question token, it considers δ as in $T(S/q_1/q_2/\ldots/q_{n-1}/q_n/\beta/\delta) \mid \beta, \delta \in A$. The method tries to find sub-tries that have a δ that is the same token as q_{n+2} (word and POS). If there are none, it tries to match δ with the next question token q_{n+3}. This continues until a match is found, or the end of the question is reached (and the EAT information present in $T(S/q_1/q_2/\ldots/q_{n-1}/q_n)$ is used). Again, if at some point multiple options are introduced, the most frequent is selected.

Another set of implementations is created that works on plain words only (instead of the combination of words and POS). They work the same as the POS-based implementations, with the difference that no tests on POS are performed. Instead of requiring tokens to have matching POS, tokens match when the words are the same.

3 Results

3.1 Data

We have applied all systems and their corresponding implementations to the annotated TREC questions [Li and Roth, 2002]. This is a collection of 5,452 questions that can be used as training data and 500 questions test data. The mean question length is 10.2 words in the training data and 7.5 words in the test data. The entire collection is already tokenized. Words and punctuation marks are separated by spaces. Each line in the files starts with a combined coarse-grained and fine-grained class (separated by a colon), that allows for testing on coarse or fine-grained set of classes and is followed by the question itself. In total, there are 6 coarse-grained classes and 50 fine-grained classes. Note that 8 fine-grained classes do not occur in the testing data. Some example questions are:

“NUM:dist”	*How tall is the Sears Building ?*
“NUM:period”	*How old was Elvis Presley when he died ?*
“LOC:city”	*What city had a world fair in 1900 ?*
“LOC:other”	*What hemisphere is the Philippines in ?*

There are some minor errors in the data that will effect the result of the structural, sequential systems described in this article.[2] In addition to an odd incorrect character, a few questions in the training data have part-of-speech tags instead of words:

Who wrote NN DT NNP NNP ” ?
What did 8 , CD NNS VBP TO VB NNP POS NN .
Why do USA fax machines not work in UK , NNP ?

In [Li and Roth, 2002], some additional problems with the data are described. Firstly, the training and testing data are gathered from different sources. In initial experiments, we found that this has quite an effect on the performance. The systems described in this article perform much better during development on the training data (using 10CV) than on the testing data.[3] However, to be able to compare our results to those in [Li and Roth, 2002], we have decided to stick with this division.

Secondly, they mention that some questions are ambiguous in their EAT. For example, *What do bats eat?* can be classified in EAT “food”, “plant”, or “animal”. They (partially) solve this by assigning multiple answers to a question. An adjusted precision metric is then needed to incorporate the multiple answers. We have not decided to do this (even though it is possible to do so), because we wanted to concentrate on the approach. The approach encompasses many different systems and the number of different systems (or parameters in the systems) would just become too big. Instead of focusing on the actual results, we want to show the validity of the structure induction approach.

3.2 Numerical results

The different systems are first allowed to learn using the training data, and after learning, they are applied to the test data. The output of the systems is compared against the correct class (as present in the test data) and the precision is computed

$$\text{precision} = \frac{\text{\# correctly classified questions}}{\text{total \# of questions}}$$

The results of all implementations applied to the coarse-grained data can be found in Table 3. To be able to compare the results, we have also computed a baseline. This simple baseline always selects the most frequent class according to the training data. This is the “HUM:ind” class for the fine-grained data and “ENTY” for the coarse-grained data. This baseline is, of course, the same for the plain words and POS tagged data. All implementations perform well above the baseline.

Looking at the results of the implementations using ABL, it shows that adding POS information to the words helps improving performance. We suspect that the POS information guides the alignment-learning phase in that the POS restricts certain alignments. For example, words that can be a noun or a verb can now only be matched when their POS is the same. However, slightly fewer regular expressions are found because of this.

The trie-based implementations do not benefit from the POS information. We think that this is because the POS information is too coarse-grained for β matching. Instead of trying to find a correct position to continue walking through the trie by finding the right word, only a rough approximation is found, namely a word that has the same POS.

Overall, it shows that the trie-based approach outperforms the ABL implementations. We do not yet have a definite answer to this difference, but we expect that it has to do with how the trie-based implementations keep track of the sequence of the words in a question, whereas the ABL-based implementations merely test if words occur in the question.

The results on the fine-grained data, given in Table 4, show similar trends. Adding POS generally improves the ABL-based system, however, this is not the case for the unhypo implementation. Again, the performance of the trie-based system decreases with POS information added.

[2]We have decided to use the original data, without correcting the errors.

[3]Similar trends can be seen when comparing results of 10CV on the training data against the results of the testing data.

Table 3: Results on the coarse-grained data

			words	POS
Baseline			0.188	0.188
ABL	hypo	default	0.516	0.682
		prior	0.554	0.624
	unhypo	default	0.652	0.638
		prior	0.580	0.594
Trie	strict		0.844	0.812
	flex		0.850	0.794

Table 4: Results on the fine-grained data

			words	POS
Baseline			0.110	0.110
ABL	hypo	default	0.336	0.628
		prior	0.238	0.472
	unhypo	default	0.572	0.558
		prior	0.520	0.432
Trie	strict		0.738	0.710
	flex		0.742	0.692

Unfortunately, it is hard to compare these results to other published results, because the classes (EATs) are often different. To be able to compare the results to at least one other system, we have decided to apply the systems to the data of [Li and Roth, 2002], which allows us to compare these results to the results of their system.

The system described in [Li and Roth, 2002] outperforms ours with a precision of 0.910 on the coarse-grained and 0.842 on the fine-grained data. However, more information is used during classification (among others, chunks and named entities). Also, it relies on the SNoW architecture. Many aspects of this architecture are well understood, which makes incorporating it in a new system easier.

To get a further idea of how the systems react to the data, we give some additional information. Each ABL-based classifier extracts roughly 115,000 regular expressions when applied to the training data and the trie that contains the set of training questions consists of nearly 32,000 nodes.

When investigating how the trie-based system works, we noticed that around 90% of the questions that performed the β replacement (15% of the total number of questions) in the strict method (on the POS tagged data) provided correct results. This indicates that β replacement in this implementations is often performed correctly. However, in the flex method on the same data, the same test shows a much lower success rate (around 65%). We think that this is due to the fewer constraints for the β substitution in the latter method, causing it to occur in more than a half of the questions.

4 Future work

The systems described here can all run without human intervention. Even though this is useful, human intervention may improve performance. This is possible by manually investigating the trie (which is a lot easier than considering all the questions) or by looking at the regular expressions generated in the ABL-based system. It is even possible to incorporate regular expressions that have been created manually. When looking at the systems in this way, they are applied as a tool to help find regular expressions that can be used to classify new questions.

Table 5: Similar tokens found by the trie-based approach

Galileo	triglycerides	calculator	Milan
Monet	amphibians	radio	Trinidad
Lacan	Bellworts	toothbrush	Logan Airport
Darius	dingoes	paper clip	Guam
Jean Nicolet	values	game bowling	Rider College
Jane Goodall	boxcars	stethoscope	Ocho Rios
Thucydides	tannins	lawnmower	Santa Lucia
Damocles	geckos	fax machine	Amsterdam
Quetzalcoatl	chloroplasts	fountain	Natick
Confucius	invertebrates	horoscope	Kings Canyon

The framework described in this article is not limited to the implementations described here. In fact, other implementations can be thought of, which we plan to do in the near future. For example, some preliminary experiments show that in the case of the ABL-based system, it may be useful to retain only the regular expressions that are uniquely associated with a EAT. Taking only these regular expressions makes handtuning of the expressions easier and because the amount of regular expressions is reduced, it makes them more manageable.

Other implementations that can be tested include those that use the non-terminal labels found by ABL or use the output of the selection learning phase of ABL (instead of the alignment learning phase only). Perhaps other grammatical inference techniques are better suited for the task described here. Alternative trie-based implementations include those using finite-state grammatical inference techniques such as EDSM or blue-fringe [Lang *et al.*, 1998].

An interesting characteristic we found in the trie-based system is the discovering of semantic relations between the replaced tokens and their β substitutes. In Table 5 we give a few (sub)sets of related tokens found.

Even thought these clearly semantically related sets of tokens can be found, many of the substitutions take place between non-semantically related tokens. However, the lack of semantic proximity between the tokens does not mean that the substitution will cause degradation in the question classification, since they do occur between tokens sharing a similar role in the question.

Similar groups of syntactic/semantic related words or word groups can be found using the ABL framework as well, as shown in [van Zaanen, 2002, pp. 76–77]. In fact, these are found using their context only. Words or word groups that tend to occur in the same contexts also tend to be used similarly. This is what is found in both systems. We expect that this knowledge will also help to build new systems that take the syntactic/semantic information into account.

Finally, we argue that the structure (in the form of a trie or regular expressions) may help finding the focus of the question. This is another part of the question analysis phase of question answering systems. Next to knowing what type of answer is expected, the system needs to know what the question is about. The learned structure may give an indication on where in the question this information can be found. For example, if the regular expression `/How far is .* from .*/` is found, the system can understand that it should find the distance between the values of the variables (`.*`).

5 Conclusion

In this article, we have introduced a new approach to question classification. Based on structure that is extracted from the training data, new questions are classified to their expected answer type.

Here, we have illustrated the approach by explaining and comparing several implementations of two systems. One system uses a grammatical inference system, Alignment-Based Learning, the other system makes use of a trie structure.

The results on the annotated questions of the TREC10 data show that the approach is feasible and both systems generate acceptable results. We expect that future systems that fall in the structure induced question classification approach (for example, based on other grammatical inference systems) will result in even better performances. Additionally, we think that the structure found by the systems can be used to find the focus of the question.

References

[Brill, 1992] Eric Brill. A simple rule-based part-of-speech tagger. In *Proceedings of ANLP-92, third Conference on Applied Natural Language Processing*, pages 152–155, Trento, Italy, 1992.

[Clément *et al.*, 1998] J. Clément, P. Flajolet, and B. Vallée. The analysis of hybrid trie structures. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 531–539, Philadelphia:PA, USA, 1998. SIAM Press.

[Geertzen and van Zaanen, 2004] Jeroen Geertzen and Menno van Zaanen. Grammatical inference using suffix trees. In Georgios Paliouras and Yasubumi Sakakibara, editors, *Grammatical Inference: Algorithms and Applications: Seventh International Colloquium, (ICGI); Athens, Greece*, volume 3264 of *Lecture Notes in AI*, pages 163–174, Berlin Heidelberg, Germany, October 11–13 2004. Springer-Verlag.

[Harabagiu *et al.*, 2000] Sanda Harabagiu, Dan Moldovan, Marius Paşca, Rada Mihalcea, Mihai Surdeanu, Răzvan Bunescu, Roxana Gîrju, Vasile Rus, and Paul Morărescu. FALCON: Boosting knowledge for answer engines. In E.M. Voorhees and D.K. Harman, editors, *Proceedings of the Ninth Text REtrieval Conference (TREC-9); Gaithersburg:MD, USA*, number 500-249 in NIST Special Publication, pages 479–488. Department of Commerce, National Institute of Standards and Technology, November 13–16 2000.

[Hermjakob, 2001] Ulf Hermjakob. Parsing and question classification for question answering. In *Proceedings of the Workshop on Open-Domain Question Answering held at the 39th Annual Meeting of the Association for Computational Linguistics (ACL) and 10th Conference of the European Chapter; Toulouse, France*. Association for Computational Linguistics (ACL), July 6–11 2001.

[Lang *et al.*, 1998] Kevin J. Lang, Barak A. Pearlmutter, and Rodney A. Price. Results of the Abbadingo One DFA learning competition and a new evidence-driven state merging algorithm. In V. Honavar and G. Slutzki, editors, *Proceedings of the Fourth International Conference on Grammar Inference*, volume 1433 of *Lecture Notes in AI*, pages 1–12, Berlin Heidelberg, Germany, 1998. Springer-Verlag.

[Li and Roth, 2002] Xin Li and Dan Roth. Learning question classifiers. In *Proceedings of the 19th International Conference on Computational Linguistics (COLING); Taipei, Taiwan*, pages 556–562. Association for Computational Linguistics (ACL), August 24–September 1 2002.

[Mitchell, 1997] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York:NY, USA, 1997.

[Pizzato, 2004] Luiz Pizzato. Using a trie-based structure for question analysis. In Ash Asudeh, Cécile Paris, and Stephen Wan, editors, *Proceedings of the Australasian Language Technology Workshop; Sydney, Australia*, pages 25–31, Macquarie University, Sydney, Australia, December 2004. ASSTA.

[TREC, 2003] *Proceedings of the Twelfth Text Retrieval Conference (TREC 2003); Gaithersburg:MD, USA*, number 500-255 in NIST Special Publication. Department of Commerce, National Institute of Standards and Technology, November 18–21 2003.

[van Zaanen and Adriaans, 2001] Menno van Zaanen and Pieter Adriaans. Alignment-Based Learning versus EMILE: A comparison. In *Proceedings of the Belgian-Dutch Conference on Artificial Intelligence (BNAIC); Amsterdam, the Netherlands*, pages 315–322, October 2001.

[van Zaanen, 2002] Menno van Zaanen. *Bootstrapping Structure into Language: Alignment-Based Learning*. PhD thesis, University of Leeds, Leeds, UK, January 2002.

[van Zaanen, 2003] Menno van Zaanen. Theoretical and practical experiences with Alignment-Based Learning. In *Proceedings of the Australasian Language Technology Workshop; Melbourne, Australia*, pages 25–32, December 2003.

[Voorhees and Buckland, 2002] E.M. Voorhees and Lori P. Buckland, editors. *Proceedings of the Eleventh Text REtrieval Conference (TREC 2002); Gaithersburg:MD, USA*, number 500-251 in NIST Special Publication. Department of Commerce, National Institute of Standards and Technology, November 19–22 2002.

[Zhang and Sun Lee, 2003] Dell Zhang and Wee Sun Lee. Question classification using support vector machines. In Charles Clarke, Gordon Cormack, Jamie Callan, David Hawking, and Alan Smeaton, editors, *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 26–32, New York:NY, USA, 2003. ACM Press.