Classifying Sentences using Induced Structure[⋆]

Menno van Zaanen, 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. In this article we will introduce a new approach (and several implementations) to the task of sentence classification, where pre-defined classes are assigned to sentences. This approach concentrates on structural information that is present in the sentences. This information is extracted using machine learning techniques and the patterns found are used to classify the sentences. The approach fits in between the existing machine learning and hand-crafting of regular expressions approaches, and it combines the best of both. The sequential information present in the sentences is used directly, classifiers can be generated automatically and the output and intermediate representations can be investigated and manually optimised if needed.

1 Introduction

Sentence classification is an important task in various natural language processing applications. The goal of this task is to assign pre-defined classes to sentences (or perhaps sequences in the more general case). For example, document summarisers that are based on the method of text extraction, identify key sentences in the original document. Some of these summarisers (e.g. [4, 8]) classify the extracted sentences to enable a flexible summarisation process.

Sentence classification is also a central component of question-answering systems (e.g. the systems participating in the question answering track of TREC[1]). Different types of questions prompt different procedures to find the answer. Thus, during a question analysis stage the question is classified among a set of predefined expected answer types (EAT). Classes can be, for instance, "number" or "location", but they can also be more fine-grained, such as "number-distance", "number-weight", "location-city", or "location-mountain".

Here, we will introduce a novel approach to the problem of sentence classification, based on structural information that can be found in the sentences. Reoccurring structures, such as *How far* . . . may help finding the correct sentence

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

[1] See http://trec.nist.gov/ for more information.

class (in this case a question with EAT "distance"). The approach described here automatically finds these structures during training and uses this information when classifying new sentences.

In the next section, the main approach will be described, as well as two different systems (with corresponding implementations) based on this idea. To get a feeling for whether the approach is feasible, we have applied implementations of the systems to real data in the context of question classification. The results published here indirectly provide insight in the approach. We will conclude with an overview of the advantages and disadvantages of these systems and finish with a discussion of future work.

2 Approach

Past approaches to sentence classification can be roughly divided into two groups: *machine learning* and *regular expression* approaches.

Grouping sentences in a fixed set of classes is typically a classification task for which "standard" machine learning techniques can be used. In fact, this has been done in the past [4, 6, 14]. However, an inconvenience of these methods is that the intermediate data produced by most machine learning techniques is difficult to interpret by a human reader. Therefore, these methods do not help to understand the classification task, and consequently the determination of the features to use by the classifier is usually reduced to a process of trial and error, where past trials do not help much to determine the optimal set of features.

Also, many features that are or have been used do not allow a description of the inherently sequential aspect of sentences. For example, bags of words, occurrence of part-of-speech (POS) tags, or semantically related words, all lose the sequential ordering of words. The order of words in a sentence contains information that may be useful for the classification.

The alternative approach, handcrafted regular expressions that indicate when sentences belong to a certain class do (in contrast to the machine learning approach) maintain word order information. As an example, one might consider the regular expression `/^How far/` that can be used to classify questions as having an EAT of "distance". This method brings fairly acceptable results when there are specific structures in the sentence that clearly mark the sentence type, as is the case with question classification into a coarse-grained set of EAT [9, 13]. The regular expression approach also 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 to manually create regular expressions that distinguish between closely related classes.

Our method aims at using the advantages of both methods based on machine learning and methods based on regular expressions. Using machine learning, patterns are extracted from the training data. The resulting patterns are easy to read by humans, and we believe they give insight about the structure of the sentences and the ordering of words therein. These patterns serve as regular expressions during the classification task. An overview of the approach is given in Figure 1.

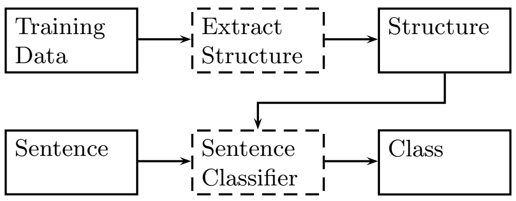

Fig. 1. Overview of the structure induction classification process

In the next two sections, we will describe two classifiers that fit into this approach. The alignment-based learning classifier uses a grammatical inference system to find structure in the sentences explicitly. This structure is converted into a set of regular expressions that are fed to the classifier. The trie classifier builds a trie structure that represents the sentences in the training corpus. It finds regularities in the trie and classifies new sentences 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 using plain text only. It has been applied to several corpora in different languages with good results [10] and it also compares well against other grammatical inference systems [12].

The underlying idea of ABL is that constituents in sentences can be interchanged. To give an example, if we substitute 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, are found and marked as such by grouping them. This is called the alignment learning phase. Table 1 shows the structure that would be learned from a sample of four sentences. Grouping is marked by parentheses.

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. However, we are not interested in this now, we only need the structure that is found. Therefore, neither clustering nor selection learning are used here and we only concentrate on the results of the alignment learning phase.

Several methods of the alignment learning phase have been described in the literature [3, 10]. Due to practical restrictions of time and space (memory) [11], we have used the suffixtree implementation [3]. 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.

Once the structure has been found, it is extracted and used in classification. Each training sentence is associated with its corresponding class, so the structure extracted from a certain sentence provides some evidence that this structure is related to the class. If a structure occurs with several classes (i.e. it occurs in multiple sentences that have different classes associated with them), the structure is stored with all classes and their respective frequencies.

The stored structures can be seen as regular expressions, so during classification, all structures can be matched to the new sentence. If a regular expression matches, its class and corresponding 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. For this, we take a look at question classification with a set of four training questions and two EAT. Consider the questions combined with the structure found by ABL and their EAT depicted in Table 1.

Table 1. Example questions with ABL structure

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

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			

The first implementation, which we will call *hypo*, takes the words in the hypotheses (i.e. the words between the brackets), turns them into regular expressions and stores them together with the corresponding EAT of the questions. The resulting regular expressions can be found in Table 2. For example, the first two questions 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 question when it is turned into a regular expression. The regular expressions extracted from the example questions can also be found in Table 2. For 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 classes of the sentence differently. Both start with trying to match each of the stored regular expressions against the sentence. If a regular expression matches, the first method (called *default*) will increase the counts of the classes of the question with the frequency that is stored with the classes of the regular expression.

The second method (called *prior*) will increase the counts of the classes of the sentence that are related to the regular expression with 1 (where the default method would increase it with the frequency count of each class).

When all regular expressions are tried, the default method selects the class 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 class with the highest overall frequency.

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 be used directly to classify new, unseen sentences. The work discussed here is an extension of the work in [7].

A trie $T(S)$ is a well-known data structure that can be used to store a set of sentences in an efficient way. It is defined by a recursive rule $T(S) = \{T(S/a_1), T(S/a_2), \ldots, T(S/a_r)\}$, where S is a set of sequences (sentences, in our case). S/a_n is the set of sequences that contains all sequences of S that start with a_n, but stripped of that initial element [2].

In each node, local information extracted during training is stored. This includes the word, class and frequency information. Since each node represents part of a unique path in the trie, frequency information is the number of sentences that use that particular node in a path in the trie.

To get a feeling for what a trie looks like, Figure 2 illustrates the trie containing the questions of Table 1 (without the bracketed structure generated by ABL). Extra information is stored per node, so for instance, the node that is reached by following the path *What* contains "DESC": 2 and "LOC": 1 as frequency information, which is similar to the information found by the unhypo system.

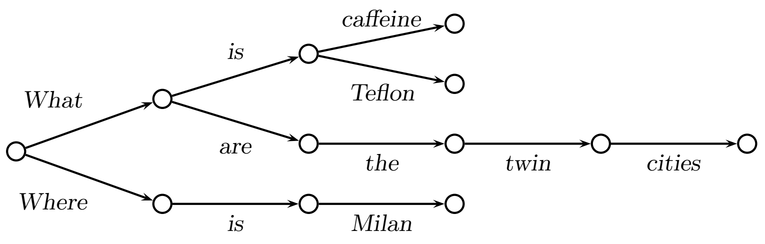

Fig. 2. The trie containing the example questions

The classification of new sentences is performed by extracting the class information stored in the trie nodes. The words of the new sentence are used to find a path in the trie. The class information can then be extracted from the final node of the path through the trie. To be more concrete, consider a sentence $S = (s_1, s_2, \dots, s_n)$ and a trie $T(S)$. The classification of the sentence is done by finding the path $T(S/s_1/s_2/\dots/s_n)$. The class information stored in the final node of the path is returned.

Notice that only with the sentences present in the training data, we can find a complete path in the trie (ending in a final node). To find a path for sentences that have not been seen in the training data, we skip non-matching words 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 words of sentence S match up to s_k, $k < n$, and the next word does not match s_{k+1}. That is, $T(S/s_1/s_2/\dots/s_{k-1}/s_k) \neq \emptyset$, and $T(S/s_1/s_2/\dots/s_{k-1}/s_k/s_{k+1}) = \emptyset$. The look-ahead process then builds a set of sub-tries $T(S/s_1/s_2/\dots/s_{k-1}/s_k/\beta/s_{k+2})$, with $\beta \in \{x|T(S/s_1/s_2/\dots/s_{k-1}/s_k/x) \neq \emptyset\}$.

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 $(s_1, s_2, \dots, s_{k-1}, s_k, \beta, s_{k+2})$ 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 word until all words in the sentence are consumed. In a sense we are replacing s_k with β. This defines a final node for the particular sentence and the class information present in that node is used to find a class.

Here, we will describe two different implementations of the classification system using a trie structure. The first uses a method called *strict*, where a word can only be replaced (if needed) by another word if s_{k+2} in the sentence matches the information in the trie. If there are several options, the one that has the highest frequency is selected. If no option is found, the search process stops at that node and the class information is retrieved from node s_k.

The second trie-based implementation, called *flex*, allows a word in the sentence to be replaced with one or more words in the trie. Initially, it works similar to the strict method. However, if no β set of sub-tries can be found, because no match on s_{k+2} can be found, it considers δ as in $T(S/s_1/s_2/\ldots/s_{k-1}/s_k/\beta/\delta)$, where β can be anything and δ should match s_{k+2}. If no such δ can be found, it tries to match δ with the next sentence token, i.e. s_{k+3}. This continues until a match is found, or the end of the sentence is reached (and the class information present in $T(S/s_1/s_2/\ldots/s_{k-1}/s_k)$ is used). Again, if at some point multiple options are introduced, the most frequent is selected.

3 Results

3.1 Data

To get a feeling of the feasibility of the structure-based classification approach, we have applied the implementations described above to the annotated TREC questions [6]. This is a collection of 5,452 questions that can be used as training data and 500 questions testing data. The mean question length is 10.2 words in the training data and 7.5 words in the testing data. The corpus contains coarse-grained and fine-grained EAT information for each of the questions. 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 affect 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 POS 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 [6], 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 ten fold cross-validation) than on the testing data.[3] However, to make the most of the limited amount of data that is available, we have decided to stick with this division.

Secondly, in [6] it is mentioned that some questions are ambiguous in their EAT. For example, *What do bats eat?* can be classified in EAT "food", "plant",

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

[3] The difference in performance of the systems displayed here is similar to that during development.

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 decided not to do this (even though it is possible to do so), because we investigate the feasibility of the approach, which is not only dependent on the results of the systems applied to a particular problem. The approach encompasses many different systems of which only a few are tested here. Instead of focusing on the actual results, we want to show the validity of the structure induction approach in general.

Part-of-speech In addition to applying the systems on the plain text, we also apply them to tokens consisting of the word combined with their POS tag. This illustrates that the algorithms in the classification framework are not just limited to plain text. Extra information can be added to the data.

We create the POS information using the Brill tagger [1]. The POS information is simply combined with the plain words. However, adjacent words that have the same POS are combined into one token. Thus, 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.

The ABL approach simply uses these complex tokens as elementary units, so for tokens to be aligned, both the word and its POS tag have to match. The trie approach is slightly modified. When searching for a matching β (i.e. when the word and POS tag do not match in the trie), the additional information of that position in the sub-tries should match. This means that in this case, the POS tag of the β nodes in the trie should match the POS tag of the word in that position in the sentence.

3.2 Numerical Results

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

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

The results of all implementations 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 coarse-grained 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 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 word sequences occur in the question.

Table 3. Results of the ABL and Trie systems on question classification

			coarse		fine	
			words	POS	words	POS
Baseline			0.188	0.188	0.110	0.110
ABL	hypo	default	0.516	0.682	0.336	0.628
		prior	0.554	0.624	0.238	0.472
	unhypo	default	0.652	0.638	0.572	0.558
		prior	0.580	0.594	0.520	0.432
Trie	strict		0.844	0.812	0.738	0.710
	flex		0.850	0.794	0.742	0.692

The results on the fine-grained data 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 when POS information is added.

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 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.

Unfortunately, it is hard to compare these results to other published results, because the data and corresponding classes (EAT) are often different. Given that [6] uses more information during classification (among others, chunks and named entities), the comparison is not reliable. We decided to compare our results to those in [14], who used the same question data to train various machine learning methods on bag-of-words features and bag-of-n-grams features. Their results ranged from 75.6% (Nearest Neighbours on words) to 87.4% (SVM on n-grams) in the coarse-grained data and from 58.4% (Naïve Bayes on words) to 80.2% (SVM on words) in the fine-grained data. Our results fit near the top on course-grained and near the middle on fine-grained data.

4 Future Work

The results on the question classification task indicate that classification based on induced structure is an interesting new approach to the more general task of sentence classification. However, there are many more systems that fall into the structure-based classification framework. These systems can be completely different from the ones described here, but modifications or extensions of these systems are also possible.

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 class. Taking only these regular expressions makes hand-tuning of the expressions easier and the amount of regular expressions is reduced, which makes them more manageable.

More complex regular expressions can be learned from the data, incorporating more information from the structure found by the system. For instance, there can be further restrictions on the words that are skipped by the systems or even systems that perform robust partial parsing using the extracted structure can be envisioned.

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 [5].

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 4 we give a few subsets of related tokens that have been found. Similar groups of syntactic/semantic related words or word groups can be found using the ABL framework as well, as shown in [10, 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, which is found in both systems. We plan to use this knowledge to place extra restrictions in the regular expressions.

Finally, we suspect that the structure (in the form of a trie or regular expressions) may help in tasks related to sentence classification, such as finding the focus of the question, which is another part of the question analysis phase of question answering systems. 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

Table 4. 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

it should find the distance between the values of the variables (`.*`). This is one area that we intend to explore further.

5 Conclusion

In this article, we have introduced a new approach to sentence classification. From the (annotated) training data, structure is extracted and this is used to classify new, unseen sentences. Within the approach many different systems are possible and in this article we have illustrated and evaluated two different systems. One system uses a grammatical inference system, Alignment-Based Learning, the other system makes use of a trie structure.

The approach falls in between the two existing approaches: “standard” machine learning techniques and manually created regular expressions. Using machine learning techniques, regular expressions are created automatically, which can be matched against new sentences. The regular expressions are human-readable and can be adjusted manually, if needed.

The results on the annotated questions of the TREC10 data, which stand in line with the current state-of-the-art results, show that the approach is feasible and both systems generate acceptable results. We expect that future systems that fall in the structure induced sentence classification framework will yield even better performance.

References

1. 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.
2. 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.

3. 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.
4. Ben Hachey and Claire Grover. Sentence classification experiments for legal text summarisation. In *Proceedings of the 17th Annual Conference on Legal Knowledge and Information Systems (Jurix 2004)*, 2004.
5. 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.
6. 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.
7. 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.
8. Simone Teufel and Marc Moens. Argumentative classification of extracted sentences as a first step towards flexible abstracting. In Inderjeet Mani and Mark Maybury, editors, *Advances in automatic text summarization.* MIT Press, 1999.
9. *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.
10. Menno van Zaanen. *Bootstrapping Structure into Language: Alignment-Based Learning.* PhD thesis, University of Leeds, Leeds, UK, January 2002.
11. 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.
12. 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.
13. 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.
14. 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.