Learning to Tag Multilingual Texts Through Observation

Scott W. Bennett and **Chinatsu Aone** and **Craig Lovell**
SRA International
4300 Fair Lakes Court
Fairfax, VA 22033
{bennett,aonec,lovellc}@sra.com

Abstract

This paper describes RoboTag, an advanced prototype for a machine learning-based multilingual information extraction system. First, we describe a general client/server architecture used in learning from observation. Then we give a detailed description of our novel decision-tree tagging approach. RoboTag performance for the proper noun tagging task in English and Japanese is compared against human-tagged keys and to the best hand-coded pattern performance (as reported in the MUC and MET evaluation results). Related work and future directions are presented.

1 Introduction

The ability to tag proper names such as organization, person, and place names in multilingual texts has great value for tasks like information extraction, information retrieval, and machine translation (Aone, Charocopos, and Gorlinsky, 1997). The most successful systems currently rely on hand-coded patterns to identify the desired names in texts (Adv, 1995; Def, 1996). This approach achieves its best performance using different hand-coded rule sets for each language/domain pair. Several of these systems have improved in ease of use, particularly in the speed of the *write pattern/evaluate performance/refine pattern* loop which plays the central role in the development process. One approach in name tagging is to assist in the creation of hand-coded rules by making it easier for the developer to mark parts of the name and its surrounding context to include in the pattern. This boosts productivity in hand-coding rules but still requires a significant amount of effort by the developer to identify key parts of the pattern. A step up from this is to determine how to generalize the rule so that it is more broadly applicable or to suggest to the developer which parts of the context have high-value for inclusion in the pattern. Nevertheless, a skilled developer with a thorough knowledge of the particular pattern language is still essential.

Our goal in developing RoboTag was to make it possible for an end-user to build a tagging system simply by giving examples of what should be tagged, rather than requiring the user to understand a pattern language. RoboTag uses a machine learning algorithm to discover features that the training examples have in common. This knowledge is used to construct a tagging procedure that can find additional, previously unseen examples for extraction.

It was important (for the confidence of our users) that the tagging procedure induced by the system be easily explained in terms of how it makes its decisions. This was one of the factors that led us to consider using decision trees (Quinlan, 1993) as a key component of the system. Other potential learning or statistical approaches for a problem like this (e.g., Neural Nets or Hidden Markov Models) did not offer this advantage. The RoboTag system is particularly well instrumented for exploration of different learning system parameters and inspection of the induced tagging procedures.

First, we discuss the overall architecture for the RoboTag system. Next, we focus on the machine learning algorithm employed for tag learning. We then present experimental results which compare RoboTag to both human-tagged keys and to the best hand-coded rule systems. Lastly, related work and future directions are discussed.

2 RoboTag Architecture

RoboTag design was motivated by our goal of developing an interactive learning system. The system had to process a large number of texts as well as provide the ability to visualize learning results and allow feedback to the learning system. To this end, RoboTag was designed as a client/server architecture. The client interface is an enhancement of a manual annotation tool. The interface works with multiple languages and includes support for both single- and double-byte coding schemes. We focus on English and Japanese in this paper. The server portion of the system performs all the document management, text preprocessing, and machine learning functions. Because it was important to facilitate interaction between the user and the learning system, it was essential to show learned results rapidly. By separating the client interface from the server which performs the learning functionality, it was possible to use fast machines for the CPU-intensive learning operations rather than relying on the user's desktop machine.

2.1 Client Interface

The client consists of a tagging tool interface written in Tk/Tcl, a cross-platform GUI scripting language. The interface, shown in Figure 1, is designed primarily to function as a tagging tool. It makes it easy for a user to mark and edit tags within multilingual texts. The tool reads and writes texts in SGML format. What distinguishes this tagging tool is that the manually tagged documents are passed back through the RoboTag server to build a tagging procedure in line with what the user is tagging. RoboTag can thus suggest what should be tagged after having received some training through observation of the user. The interface has been augmented with several displays that allow for a thorough investigation of the learned tagging procedure. These include graphical displays of the induced logic for tagging (cf. Figure 2), graphical displays of tagging accuracy (i.e. precision and recall), and the ability to inspect the examples from the texts that justify the induced tagging procedure.

2.2 Server

The RoboTag server performs the tag learning functions. It manages the training and testing files, extracts features, learns tagging procedures from tagged training texts, and applies them to unseen test texts. Each RoboTag client invokes its own instance of the server to handle its learning tasks. There can be multiple servers running on the same machine, each independently handling a single client's tasks. The RoboTag server receives commands from the client and returns learning results to it. During this dialogue, the server maintains intermediate results such as learned tagging procedures, texts that have been preprocessed for learning or evaluation, and state information for the current task. This includes the parameter settings for the learning algorithm, feature usage statistics, and preprocessor output. The client connects to the RoboTag server on a network using TCP/IP. There is a well-defined interface to the server so it can act as a learning engine for other text handling applications as well.

Examples of server commands include:

1. Process a text for training or testing
2. Learn a classifier[1] for a tag
3. Evaluate a learned classifier on a text
4. Load a previously learned classifier or save one for future use
5. Change a learning parameter
6. Enable or disable a lexical feature

3 Learning to Tag

RoboTag must learn to place tags of varying types within the text. This means placing an appropriate SGML begin tag like <PERSON> prior to a person's name in the text and following the person's name with an SGML end tag like </PERSON>. In this paper, in order to compare with other name tagging system results as reported in the Message Understanding Conference 6 (MUC-6) (Adv, 1995) and the Multilingual Entity Task (MET) (Def, 1996), we will be tagging people, places, and organizations. RoboTag provides for learning other types of tags as well.

For each tag learning task, RoboTag builds two decision trees - one to predict begin tags and one to predict end tags. The results of these classifiers are then combined using a tag matching algorithm to yield complete tags of each type. A tag post-processing step resolves overlapping tags of different types using a prioritization scheme. Altogether, these make up the learned tagging procedure.

In this section we describe RoboTag's decision tree learning, learning representation, learning parameters, the tag matching algorithm, and post-processing.

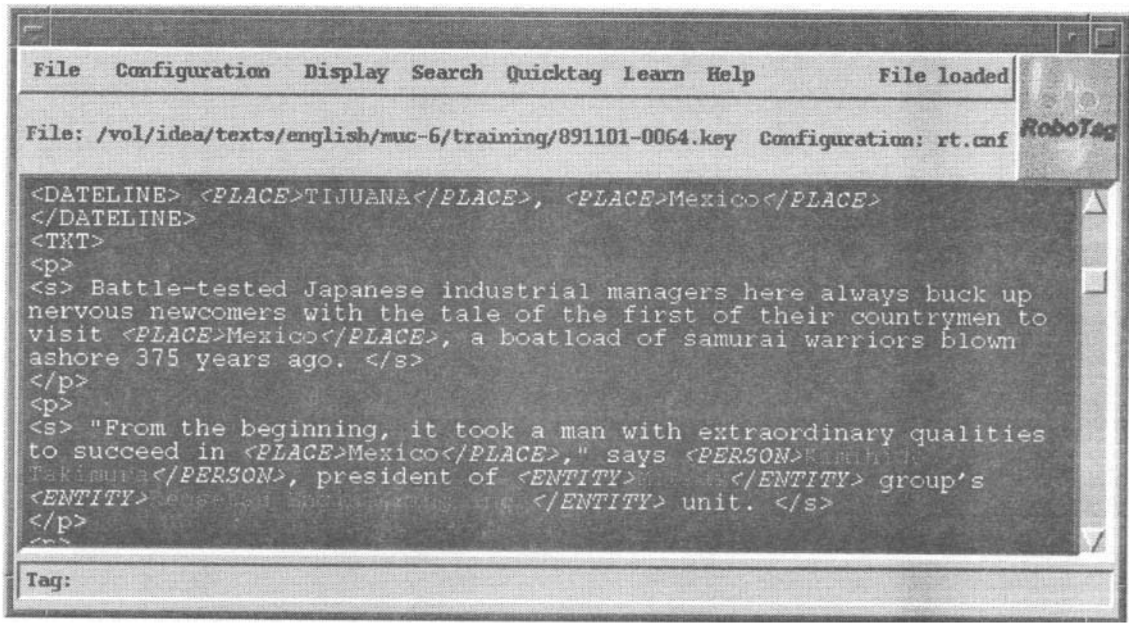
3.1 Decision Tree Learning

RoboTag learns decision tree classifiers that predict where tags of each type should begin and end in the text. The decision trees are trained from texts which have already been tagged manually.

For learning the tag begin/end classifiers, we build decision trees using C4.5 (Quinlan, 1993).[2] C4.5 is used to learn decision tree classifiers which distinguish items of one class from another based on attributes of the training examples. These attributes are referred to as *features*. In using a decision tree for classification, each node indicates a feature test to be performed. The result of the test indicates which branch of the tree to take next. Ultimately, a leaf node of the tree is reached which specifies the classification result. To produce our decision trees, an information theoretic criteria called information

[1] RoboTag uses decision tree classifiers as part of the learned tagging procedure. They will be discussed in the next section.

[2] C4.5 has been specially adapted to work directly on our preprocessor-produced data structures for more efficient operation rather than through data files which is the normal mode of operation.



```
File: /vol/idea/texts/english/muc-6/training/891101-0064.key  Configuration: rt.cnf
```

```
<DATELINE> <PLACE>TIJUANA</PLACE>, <PLACE>Mexico</PLACE>
</DATELINE>
<TXT>
<p>
<s> Battle-tested Japanese industrial managers here always buck up
nervous newcomers with the tale of the first of their countrymen to
visit <PLACE>Mexico</PLACE>, a boatload of samurai warriors blown
ashore 375 years ago. </s>
</p>
<p>
<s> "From the beginning, it took a man with extraordinary qualities
to succeed in <PLACE>Mexico</PLACE>," says <PERSON>Kimihide
Takimura</PERSON>, president of <ENTITY>Mitsui</ENTITY> group's
<ENTITY>Kensetsu Engineering Inc.</ENTITY> unit. </s>
</p>
```

Figure 1: RoboTag Interface

gain ratio is used to measure, at each step of tree construction, which feature test would best distinguish the examples on the basis of their class. The simplest classification problem involves learning to distinguish positive and negative examples of some concept. In our case, this means characterizing text positions where a tag should begin or end from text positions in which it should not.

In order to extract learning features, the RoboTag server employs a preprocessor plug-in for each language it operates with. This preprocessor performs tokenization, word segmentation, morphological analysis, and lexical lookup as necessary for each language. The preprocessor produces output in a well-defined format across languages which the server uses in carrying out the learning. For instance, in processing Japanese, RoboTag may use features which are uniquely Japanese but may not be present in English, or *vice versa*. Table 1 shows some of the features used by RoboTag for learning.

Figure 2 shows a screen shot of a portion of a decision tree trained to produce begin tags. One of the leaf nodes of the tree has been selected producing a window which shows person names in context as classified at the leaf. The last test in the branch prior to the shown window tests to see if the word prior to the current word is a person title (like "President," "Secretary," or "Judge" when a decision is being made about whether to start a name with "Reagan," "Robert," or "Galloway" respectively). The screen shot goes on to show that if the previous word is not a person title, the system consults the 2nd word prior to the candidate begin tag to see if it is an organization noun prefix (such as "bank", "board", or "court").

3.2 Learning Representation

C4.5 represents training examples as fixed length feature vectors with class labels. The goal is to learn to predict the class label from the other features in the vector. In our case this means learning to label tokens as begin or end tags from the token's lexical features. When RoboTag processes a tagged training text, it creates labeled feature vectors (called *tuples*) from the preprocessor data. One tuple is created for each token in the text, with the label TRUE or FALSE. If we are learning a tree to predict begin tags, the label is TRUE if the token is the first token inside an SGML tag we are trying to learn, and false otherwise. Similarly for end tags, the tuple is labeled TRUE if the token is the last token in a training tag and false otherwise.

A single token usually does not contain enough information to decide whether it makes a good tag begin or end. Features from the surrounding tokens must be used as well. To create a tuple from a token, RoboTag collects the preprocessor features for the token as well as its immediate neighbors. How many neighboring tokens to use is determined by a *radius* parameter, as will be discussed in Section 3.3. A radius of 1 means the current token and both the previous and next tokens will be part of the tuple (1 token in each direction).

To fill in the tuple values, RoboTag calls on the preprocessor as a feature extractor. Each position in the tuple's feature vector holds a value from a preprocessor field. RoboTag can use whatever lexical and token-type features that the preprocessor pro-

Table 1: Features Used in Learning

Features	Examples
Token Type	lower, upper, cap, hiragana, kanji, katakana
POS	adj, adv, aux, conj, det, n, prep, pro, v
Location	continent, country, province, city
Semantic Type	first name, corporate designator, title

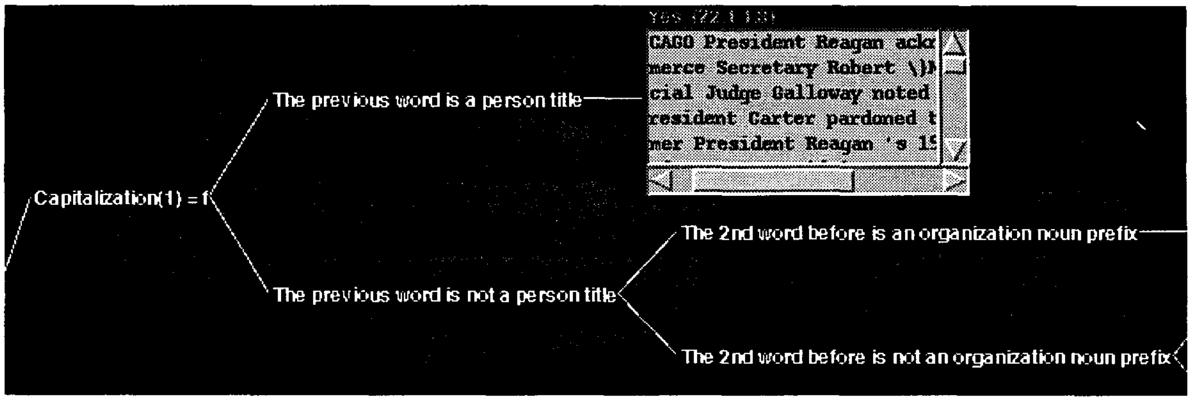

Capitalization(1) = t
The previous word is a person title
The previous word is not a person title
Yes (22.1 13)
CAGO President Reagan ackr
merce Secretary Robert \)N
cial Judge Galloway noted
resident Carter pardoned t
mer President Reagan 's 19
The 2nd word before is an organization noun prefix
The 2nd word before is not an organization noun prefix

Figure 2: Part of a RoboTag Begin Tag Classifier

vides. In this way the preprocessor forms the background knowledge for the target language. Once the training texts have been represented as tuples, the learning process can begin.

3.3 Learning Parameters

There are several parameters to RoboTag that affect tagging performance. Below are descriptions of some of the parameters. The Experiments section discusses the settings that produced the best results for each task.

- **Radius**: This controls the number of tokens used to make each training tuple. A higher radius gives the decision tree algorithm more contextual information in deciding whether a token makes a good begin or end tag.
- **Sampling Ratio**: Creating one tuple from each token in a text leads to many more negative training examples than positive, since only the tokens at the beginning (or end) of a tag generate positive training tuples. Every other token forms a negative example; a place where a tag did not begin or end. Too many negative examples can hurt learning accuracy by making the system too conservative. In some extreme cases, this can lead to decision trees that never predict a tag begin or end no matter what the input. The sampling ratio is the ratio of negative to positive examples to use for training. All of the positive examples are used, and negative examples are chosen randomly in accordance with this parameter. What is interesting about the Sampling Ratio is that it allows recall to be traded off for precision directly. Increasing the sampling ratio gives the learning system more examples of things that should not be tagged, reducing the number of false positives which increases precision. Making the decision trees more conservative in this way can also lower recall. Finding a balance of precision and recall by tuning this parameter is essential for best results.
- **Certainty Factor**: This parameter affects decision tree pruning, a process used to simplify learned decision trees. Pruning helps reduce over-fitting of training data and improves classification accuracy on unseen examples. This parameter takes values between 0 and 1, with lower values meaning more pruning.

3.4 The Matching Algorithm

When tagging a text, RoboTag evaluates the learned decision tree classifiers on the new text to produce a list of potential begin and end tags for each tag type. These lists are produced independently, and there may be many ways to pair begin and end tags together. For each begin tag found there may be several plausible end tags that could pair with it (and *vice versa*). The matching algorithm must decide the best possible pairing of the begin and end tags for each tag type.

Each potential begin and end tag produced by the decision tree also has a *confidence* rating, a number between 0 and 1 estimating the chance of correct classification. A scoring function is used to evaluate the relative merits of different sets of pairings. In addition to the confidence ratings for the tags, the scoring function makes use of statistical measures like the mean and standard deviation of the tag length in the training examples. The matcher can be biased to prefer tags longer, shorter, or closest to this mean length.

Considering all possible begin/end tag pairings quickly becomes intractable as the number of potentially interacting tags increases. Therefore, the first step in the matching process seeks to divide the text up into a set of non-interacting sections.

Each time a begin/end pair is made, any begin or end tags between the pair cannot be used (or the resulting tags would overlap). This means that each pair could preclude other possible matches. The text is divided into sections by observing which tags can possibly affect other tags. The mean distance, standard deviation, and match threshold determine the distance interval within which the matcher searches for tag pairs. If two tags are far enough apart, they can be matched independently without fear of one pairing precluding another. These boundary points in the text are found first. Then each independent section is searched separately for tag pairings. The best pairing set for a section maximizes the sum of the scores for each pair in the section.

There are three parts to the scoring function for a pair. The first is the confidence with which the begin tag tree classifies the token as a good begin tag. The second component is the end tag tree confidence. The last part is a distance score, which is calculated from the tag length, mean distance, and match length preference. Each of the three length preferences (longest, shortest, or closest to mean distance) uses an appropriate bias to the way in which these inputs are combined.

3.5 Tag Overlap Resolution

Because the tag matching algorithm only ensures non-overlapping tags within each tag type, it is possible to have cases of embedded tags of different types (like tagging "Boston" as a location within the tag for "Boston Edison Company"). To resolve these cases, RoboTag uses a static tag priority scheme. For proper noun tagging the priority order from highest to lowest is person, entity, place. We do not currently learn the tag priorities although this is a logical extension to the learning technique.

4 Experiments

We set up experiments on English and Japanese name tagging using the same texts that were used for the named entity task of the MUC-6 and MET competitions. In this way, we can most easily compare RoboTag performance against a variety of other name tagging systems.

4.1 English Results

For English, the MUC-6 Wall Street Journal corpus was used. RoboTag was trained with 300 training texts and proceeded to automatically tag the 30 blind test texts. The scores on the test set are shown in the Table 2. For each tag type, the table gives the total number of tags of that type present in the training and testing sets and the recall, precision, and F-Measure[3] as measured on the test set. Overall totals are given at the bottom of the table.

The best system in the MUC-6 named entity task, using hand-coded rules, returned F-Measures of 98.50 for person, 96.96 for place, and 92.48 for entity as shown in Table 3 (Krupka, 1995) .

We found that RoboTag's best English results were obtained with a sampling ratio of 10, a radius of 2, and certainty factors of 0.75 for pruning for all the tag types.

4.2 Japanese Results

In Japanese, the MET corpus of press-conference related texts from Kyodo News Agency was used in the experiment. A training set of 300 texts was used with a blind test set of 99. RoboTag scores on the test set are reported in Table 4.

The best system on the MET task, utilizing hand-coded rules, produced F-Measures of 95.37 for person, 93.43 for place, and 86.90 for entity (cf., Table 5) while the second place system posted 78.54 for person, 84.00 for place, and 79.25 for entity. RoboTag would have ranked 2nd among the MET systems on the Japanese entity task.

Sampling ratios for our best Japanese results were 35, 15 and 10 for person, place, and entity. For all three tags we used a radius of 2 and certainty factors of 0.65 for pruning.

5 Related Work

Vilain and Day (Vilain and Day, 1996) report on an approach which learns and applies rule sequences for the name tagging task (based on Eric Brill's rule sequence work (Brill, 1993)). It uses a greedy algorithm to generate and apply rules, incrementally refining the target concept. They report their best precision/recall results for machine-learned rules on the MUC-6 task with equivalent F-Measures[4] of 78.50

[3]F-measure is calculated by:

$$F = \frac{(\beta^2 + 1.0) \times P \times R}{\beta^2 \times P + R}$$

where P is precision, R is recall, and β is the relative importance given to recall over precision. In this case, β = 1.0 as used in MUC-6 and MET.

[4]The F-Measure formula they report seems to be in error and they reported with a β of 0.8. For comparison, we used the standard F-Measure formula with a β of 1 as reported above.

Table 2: RoboTag English Results

Tag Type	# Training	# Testing	Testing Recall	Testing Precision	F-Measure
Person	1978	372	93.5	95.9	94.7
Place	2495	110	95.5	89.7	92.5
Entity	3551	448	79.7	83.4	81.5
Total	8024	930	87.1	89.2	88.1

Table 3: Best MUC-6 English Results

Tag Type	# Poss	Recall	Precision	F-Measure
Person	373	98	99	98.5
Place	110	99	95	96.96
Entity	447	91	94	92.48
Total	930	94.8	96.1	95.4

Table 4: RoboTag Japanese Results

Tag Type	# Training	# Testing	Testing Recall	Testing Precision	F-Measure
Person	1081	346	81.0	89.9	85.2
Place	1960	756	84.5	88.7	86.6
Entity	1958	596	77.1	80.7	78.8
Total	4999	1698	81.2	86.1	83.6

Table 5: Best MET Japanese Results

Tag Type	# Poss	Recall	Precision	F-Measure
Person	346	92	99	95.37
Place	756	91	96	93.43
Entity	596	84	90	86.90
Total	1698	88.8	94.5	91.5

for person, 74.35 for place, and 82.81 for entity. Our English score is significantly better, especially for the person and place tasks. Because their Japanese results were not reported we cannot compare our Japanese performance.

Gallippi (Gallippi, 1996) presents an approach to tag classification using decision trees. Hand-coded rules are employed to delimit proper nouns within the text. Each proper noun is then classified into an appropriate type (e.g., person, entity, place) using decision trees (ID3), an easier task than also learning to place tags. It is also less general to rely on hand coded rules for a significant part of the tagging task.

Bikel *et al.* (Bikel et al., 1997) report on Nymble, an HMM-based name tagging system operating in English and Spanish. Nymble performs well, turning in F-measures of 90 and 93 respectively in Spanish and English on the MUC-6 task. These scores were achieved using 450,000 words of tagged text, 3 times the size of the 150,000 word training set used for the RoboTag experimental results reported here. Bikel reports that moving from 100,000 to 450,000 training texts yielded a 1-2% improvement. A direct comparison with Nymble on particular tag types is not possible because only the overall F-measure is reported for the MUC-6 task. In these experiments we only trained and tested on person, place, and entity tags. If we use RoboTag with our hand-coded rules for dates and number, the overall F-measure on the MUC-6 English task is 90.1.

6 Future Directions

There are a number of ways in which RoboTag performance could be improved. Perhaps the most obvious enhancement to our representation involves giving the learning system the actual text of the token in the feature vector. Currently, each tuple contains the preprocessor information for a window of tokens in the text, but the actual token text is not available to the learning. The decision trees can refer to classes of words by their lexicon features, but not individual words themselves. Adding this capability would allow performance improvement especially in cases where lexicon data is sparse. Using words as features is related to the idea of automatic word list modification. This would allow RoboTag to actually reconfigure its knowledge base of word lists and propose new features. This is one way that RoboTag could adapt to new extraction domains.

Unlike some of the name tagging systems RoboTag is being compared to, RoboTag has no alias generation facility. By generating an alias from a recognized name, a system can scan for that alias (e.g., a company's acronym or an individual's first name) in order to improve the likelihood of identifying it. It would be straightforward to add such an alias capability to RoboTag.

Another accuracy enhancement is to improve the tag matching algorithm. RoboTag does not currently use the lexical features of the tokens during the match process. The scoring function takes into account tag length and decision tree confidence values only. Many of the errors RoboTag makes come from the matching algorithm where the decision trees correctly predict tag begins and ends but the wrong tag pairings are chosen. Making the matching algorithm sensitive to lexical features should help correct this.

Although, for comparison with other systems, we have presented traditional batch-mode learning results here, one of RoboTag's strengths is in its interactivity. We believe that allowing the user to give direct feedback to the learning system is key to rapidly addressing new extraction tasks. We plan to do further experiments which address how the use of this directed feedback can result in rapidly learned tagging procedures utilizing fewer tagged texts.

Finally, our experiments have focused on proper name tagging, but RoboTag is not limited to this. We are planning to explore additional tagging tasks besides names in multiple languages such as Chinese, Thai, Spanish as well as English and Japanese.

7 Summary

RoboTag is a multilingual text extraction system that automatically learns to tag texts by observing its users. Decision trees are learned to predict where users begin and end tags. These are combined with a matching algorithm to produce complete tags. RoboTag is flexible in its ability to work with multiple languages. We have shown RoboTag performance to be competitive with hand-coded pattern-based systems in very different languages like English and Japanese.

References

Advanced Research Projects Agency. 1995. *Proceedings of Sixth Message Understanding Conference (MUC-6).* Morgan Kaufmann Publishers.

Aone, Chinatsu, Nicholas Charocopos, and James Gorlinsky. 1997. An Intelligent Multilingual Information Browsing and Retrieval System Using Information Extraction. In *Proceedings of the Fifth Conference on Applied Natural Language Processing.*

Bikel, Daniel M., Scott Miller, Richard Schwartz, and Ralph Weischedel. 1997. Nymble: a High-Performance Learning Name-finder. In *Proceedings of the Fifth Conference on Applied Natural Language Processing.*

Brill, Eric. 1993. *A Corpus-based Approach to Language Learning.* Ph.D. thesis, University of Pennsylvania.

Defense Advanced Research Projects Agency. 1996. *Advances in Text Processsing: TIPSTER PROGRAM (Phase II).* Morgan Kaufmann Publishers.

Gallippi, Anthony. 1996. Recognizing Names Across Languages. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING).*

Krupka, George. 1995. SRA: Description of the SRA System as Used for MUC-6. In *Proceedings of Sixth Message Understanding Conference (MUC-6).*

Quinlan, J. Ross. 1993. *C4.5: Programs for Machine Learning.* Morgan Kaufmann Publishers.

Vilain, Marc and David Day. 1996. Finite-state and phrase parsing by rule sequences. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING).*