Optimizing the subtasks in the double classification approach to Information Extraction

Viktor Pekar and **Richard Evans**
Research Group in Computational Linguistics
HLSS, University of Wolverhampton
MB114 Stafford Street
Wolverhampton, WV1 1SB, UK
{v.pekar,r.j.evans}@wlv.ac.uk

Abstract

In Information Extraction, a very common task is to extract facts about a single event or entity from an entire document such as a personal homepage, a job or a seminar announcement. The double classification method approaches this task with two automatic classifiers. The first one classifies larger document fragments to roughly indicate which of them are likely to contain template fillers. The second classifies text tokens inside promising fragments to more precisely pinpoint the filler. In this study we show how the effectiveness of the method can be considerably increased by optimizing the task difficulty of each of the classifiers. We then consider the related problem of identifying the best filler per template field among all the text tokens labeled as positive instances of that field by the second classifier. We present a method to delimit a token or sequence of tokens among them as the best filler for the field.

1 Introduction

A common information extraction (IE) task is to extract facts about a single event or entity from an entire document such as a person's name and contact details from her home page. Correspondingly, for each IE task, there is one template to be filled for each document. Previous research has developed a range of IE methods based on automatic classification techniques to address this kind of task (e.g., (Freitag 98), (Soderland 99), (McCallum *et al.* 00), (Califf & Mooney 98), (Chieu & Ng 02)).

The double classification method uses two different automatic classifiers to extract information. The method is inspired by the observation that when humans need to extract some facts from a document they scan it quickly and only read closely those parts that look most relevant. In a similar manner, the double classification method uses the first classifier to identify document fragments that are likely to contain template fillers. The second classifier classifies tokens inside the promising fragments to more precisely pinpoint the filler. The study by (Sitter & Daelemans 03) has shown that this is a very promising approach to pursue. By first performing classification of fragments, the unbalanced data problem at the token level is greatly reduced, i.e. the fact that a document contains much fewer positive examples of field fillers than negative ones. De Sitter and Daelemans' study demonstrated that the method is both more efficient and effective than methods that extract template fillers by examining the immediate context of each token in the text.

This paper examines the idea that the effectiveness of the method can be increased by carefully choosing the classification problems for the two classifiers. We show that by using an appropriate fragment size and applying thresholding and instance selection techniques, the token-level classifier is able to locate template fillers more accurately than the original method proposed by (Sitter & Daelemans 03).

Another contribution of the paper is the proposal of a new method for accurate identification of one single filler for a field which may consist of a single token or a sequence of tokens. The method identifies such fillers in the output of the double classification method. It takes advantage of such evidence for the best filler as the relative position of tokens labeled as positive by the second classifier, the frequency of the token sequences, and the frequency of their subparts.

The paper is organized as follows. The next section discusses the double classification method in more detail, presents the techniques that we investigate in order to adjust the two classification problems and proposes an algorithm for the identification of the best fillers in the output of the method. Section 3 describes the settings used for experimental evaluation. In Section 4, we present and discuss the results of the evaluation. Finally, in Section 5, we summarize the results and draw conclusions.

2 The Double Classification Method

2.1 Task Definition

The IE procedure performed by the double classification method can be formally described as follows. Suppose we have a corpus annotated in terms of a predefined IE template, i.e., certain text tokens (words and punctuation) have a tag signifying that they instantiate a field of the template. The corpus is randomly divided into a training set of documents D_{TR} and a test set of documents D_{TS}. The first step is to split documents in D_{TR} and D_{TS} into sets of document fragments (say, sentences or paragraphs) F_{TR} and F_{TS}, respectively.

At each classification stage, n binary classifiers are

built, one for each template field s_i. At the fragment classification stage, classifier C_1 is learned from F_{TR}. Positive instances are fragments, which contain at least one token annotated as s_i. Features for representing an instance are all tokens found inside the fragment. C_1 is evaluated on F_{TS}. Its output is F_{out}, the set of fragments that it has labeled as positive.

At the second stage, classifier C_2 is learned only from those fragments in F_{TR} which contain tokens annotated as a filler for s_i. Positive instances here are tokens annotated as s_i, negative ones are those that do not have any tags or have tags other than s_i. Features are tags and tokens appearing within a certain window around the token. C_2 is evaluated on tokens contained in F_{out}. The output from C_2 is a list of tokens W_{out} which it has labeled as positive instances of s_i.

In computing evaluation metrics for the method as a whole, true positives are those tokens in W_{out} that have been manually annotated as positive, false positives are tokens in W_{out} that have not been manually annotated as positive, and false negatives are tokens have been manually annotated as positive, but did not make it into either F_{out} or W_{out}.

2.2 Adjusting the Task Difficulty

We hypothesize that the effectiveness of the method greatly depends on how the difficulty of the entire IE task (i.e. the search space in which template fillers should be found) is distributed between the two classifiers. We would like to find a balance that will avoid two kinds of situations leading to poor overall results:

1. C_1 achieves high precision, but low recall. The search space for C_2 is greatly shrunk, but many relevant tokens are missing from it.

2. C_1 achieves high recall, but low precision. Most of the relevant tokens do appear at the second stage, but there are also very many irrelevant ones and the search space for C_2 is too large.

In this paper, we examine whether or not the factors described in Sections 2.2.1 to 5 can help in the discovery of this optimal balance:

2.2.1 Thresholding

The most suitable balance is not simply the most balanced precision/recall ratio at C_1. It may depend on the nature of the documents at hand, namely on how indicative of template fillers: (1) larger contexts of tokens (e.g., paragraphs) and (2) their local contexts (e.g., neighboring tokens) are relative to each other. If, for example, it is easy to recognize fillers in their local contexts, but larger contexts are difficult to distinguish as relevant or irrelevant, one should be cautious about classifications at C_1 and aim to maximize C_1's recall in order to increase the number of promising tokens passed on to C_2. If, on the contrary, larger contexts are highly indicative of template fillers, then it makes sense to try to narrow down the search space for C_2 by preferring greater precision on the part of C_1.

Thresholding is a technique that allows one to boost either precision or recall by looking at the confidence score of the classifier, such as the class membership probability output by Naïve Bayes classifiers or the distance in the vector space output by k nearest neighbors and Support Vector Machine classifiers. Various thresholding techniques exist (see (Sebastiani 02)). In this study we prioritize recall by determining the average confidence score for false negatives on held-out data and during proper testing we retrieve all instances above that threshold. To increase precision, we find the average confidence score for false positives and during testing treat all instances below that threshold as negative even if the classifier assigns them the positive label.

2.2.2 Fragment Size

The size of the fragments may have a strong effect on how many relevant and irrelevant tokens will be passed from C_1 to C_2. If a document is split into many smaller fragments, such as lines, classifying them will be more difficult but this will allow for the greatest reduction of the search space for C_2. If instead one chooses larger fragments like paragraphs, relevant ones will be found with greater ease, but C_2 will suffer more from the unbalanced data problem.

Again, the best solution is not necessarily to simply choose average sized fragments; the usefulness of the fragments depends on the specific characteristics of the documents.

Note that thresholding and fragment size are orthogonal factors, so that their combination may maximize the desired effect and at the same time compensate for one another's drawbacks. For example, aiming for good recall of smaller fragments may lead to better results overall than increasing precision in the classification of larger fragments, and vice versa.

2.2.3 Selection of Instances

As in the work described in (Sitter & Daelemans 03), we also consider whether or not the problem of too many negative instances can be alleviated by performing instance selection. In order to build a more effective classification model, we train it on data from which some negative instances have been removed so that a certain desired proportion between negative and positive instances is achieved. After the model is learned from the balanced training data, it is evaluated on the unbalanced test data.

In this study we look at how instance selection interacts with the other two parameters. In particular, we wish to find out whether the problem of increased search space resulting from maximized recall or from using larger fragments can be remedied by performing instance selection.

2.3 Identifying the Best Filler

The double classification method tries to find all occurrences of a filler in the document. Obviously, this

task is difficult and seldom error-free. In many situations, however, extracting all field instantiations is not necessary, since the template field has to be filled with one single filler. One possibility to perform this is to choose out of all candidates the one that the token-level classifier extracted with the greatest confidence. However, this approach will not be very helpful in the case when fillers consist of multiple tokens, as it may easily select one part of the filler, but miss out others.

We propose a new method to identify the best filler for a template field. In addition to the classifier confidence score, it incorporates information about whether the positively labeled tokens make up sequences, the count of these sequences, and the use of general surface constraints on the appearance of the filler.

The algorithm (see Algorithm 1) consists of two major steps. The first step (lines 1-7) is to extract N, a set of all possible token ngrams from C_2's output and compute the initial score for each n. The ngrams are uninterrupted sequences of tokens labeled as positive instances of a template field. Those ngrams are eliminated that consist of tokens bearing no content such as stopwords and punctuation (line 4). Semantically empty tokens are also removed from the beginnings and ends of the ngrams (line 5). The initial score for n is computed as the sum of the weights of its occurrences, where the weight of each occurrence n_{occ} is the average classifier score C of its constituents (line 6).

To illustrate with an example, consider the hypothetical output from the classifier in Figure 1 (the first column shows the number of the token in the document, the second the token itself, the third the label assigned by the classifier, and the last the classifier confidence score). The algorithm will first extract the ngrams "by John Doe.", "John", and "Doe". Stripping stopwords and punctuation at the beginning and end of each of them, three ngrams will be obtained: "John Doe", "John", and "Doe". Their initial scores will be computed as follows:

$score(\text{"}John\ Doe\text{"}) = (0.75+0.5)/2 = 0.625$
$score(\text{"}John\text{"}) = 0.35$
$score(\text{"}Doe\text{"}) = 0.7$

```
...
18   authored   NIL      0.5
19   by         AUTHOR   0.5
20   John       AUTHOR   0.75
21   Doe        AUTHOR   0.5
22   .          NIL      0.5
...
44   writes     NIL      0.6
45   John       AUTHOR   0.35
46   .          NIL      0.45
...
72   John       NIL      0.3
73   Doe        AUTHOR   0.7
74   ,          NIL      0.9
...
```

Figure 1: Example of the output from C_2.

Data: a list of positively classified document tokens T, each attached with a classifier confidence score C
Result: a list of token ngrams ranked according to their relevance as template fillers

1 Extract a set of token sequences S from T;
2 Create a set of unique token ngrams N by extracting all subsequences from each $s \in S$;
3 **for** *each* n *in* N **do**
4 | discard n, if it contains only punctuation or stopwords;
5 | remove punctuation and stopwords in the beginning and end of n;
6 | $score(n) = \sum_{n_{occ} \in n} \frac{1}{length(n_{occ})} \sum_{i \in n_{occ}} C(i)$;
7 **end**
8 **for** *each* n *in* N **do**
9 | discover N' in N such that n' is a subsequence of n;
10 | **for** *each* n' **do**
11 | | $score(n)$ += $score(n') \times \frac{length(n')}{length(n)}$;
12 | **end**
13 **end**
14 Rank N according to *score*;

Algorithm 1: The algorithm for identifying the best filler per template field.

The second step (lines 8-13) is to add further weight to those ngrams whose subsequences exist in N. Thus, the final score for "John Doe" will be increased by the initial scores of its subsequences "John" and "Doe" appearing as distinct ngrams, each weighted by the proportion of its length to the length of the greater ngram, i.e. by 0.35*0.5 + 0.7*0.5 = 0.525. In this way, the algorithm aims to further take into account those cases, when only a part of a relevant ngram has been labeled positively by the classifier.

As it is reliant upon the count of the ngram, the method may have important interaction with the particular thresholding and fragmentation methods used. Specifically, we hypothesize that the best filler identification works best with double classification settings that achieve the greatest recall while maintaining high precision.

3 Evaluation

3.1 Experimental Task

The documents we would like to extract information from are web pages describing NLP resources including software (part-of-speech taggers, parsers, various corpus tools) and data (evaluation corpora and datasets, frequency lists, gazetteers). The IE template consists of the following fields: NAME, CREATOR, AREA (application area), TGTLANG (target language), PLATFORM, PROGLANG (programming language), and EMAIL (contact email). All the fields take single fillers, except TGTLANG and PLATFORM. Some slots are mandatory

(e.g., NAME), while others are not (e.g., TGTLANG). It should be emphasized that although some of the fields are filled by a closed class of words (e.g., PLATFORM), the IE method is a machine learning procedure that extracts fillers by examining only the context of tokens in the documents.

3.2 Data

The evaluation is carried out on 100 web pages that had been manually downloaded using the link collection on the topic at the Language Technology World web site[1]. The documents are preprocessed in the following steps:

Irrelevant HTML code (e.g., tags for images, forms, various scripts) are removed. The HTML structure is standardized and converted to XML. The documents were tagged for paragraph and sentence boundaries, parts-of-speech and syntactic chunks using the *LT Chunker* program (Mikheev 96).

3.3 Classification Method

At C_1 each fragment was represented as a feature vector, where features corresponded to the tokens found in it. All words were stemmed, stopwords and words appearing in less than 5 different fragments in the entire corpus were discarded. At C_2, to represent each token t, the following features were used:

- `token_itself`: the string corresponding to t
- `tags_itself`: XML tags (layout, PoS, phrase chunking tags) inside which t appears
- `token_before`: the token directly before t
- `tags_before`: XML tags on the token before t
- `token_after`: the token directly after t
- `tags_after`: XML tags on the token before t
- `token_window`: the tokens appearing within the context window of 2 around t
- `tags_window`: all XML tags appended on the tokens within the context window.

In the experiments we used the WEKA implementation of the multinomial Naïve Bayes learner (Witten & Frank 99). To assess the accuracy of classifications, we use 10-fold cross-validation, computing precision, recall and F-measure for each field and then averaging the results.

4 Results and Discussion

4.1 Fragmentation Method

We experimented with four types of fragments: Sections (*Sec*), Paragraphs (*Par*), Sentences (*Sent*) and Lines (*Lin*). Table 1 characterizes each type of fragments.

[1] http://www.lt-world.org

	Sec	Par	Sent	Lin
Tokens per fragment	68.5	18.8	9.4	7.3
Fragments per doc	15	54.2	101.6	142.1

Table 1: The average size of fragments and the number of fragments per document for the four fragmentation methods.

	No thresholding	Boosted P	Boosted R
Sec	0.02	0.03	0.01
Par	0.05	0.06	0.04
Sent	0.81	0.08	0.77
Lines	0.09	0.1	0.08

Table 3: Search space at C_2: the proportion of positive and negative training instances for different fragmentation methods and thresholding settings.

Table 2 describes the effectiveness of classifications at both levels (C_1 and C_2) resulting from the use of each fragmentation method (*Sec*, *Par*, *Sent*, *Lin*). The best results across fragmentation methods are shown in bold. Column 1 in Table 3 characterises the search space for each fragmentation method as the corresponding proportion of positive and negative instances at the token level.

We see that at C_1 larger fragments do indeed result in an easier classification task: the highest effectiveness at the first stage is achieved for the *Sec* and *Par* methods. Looking at C_2, we notice that the proportion of positive instances increases as the fragment size decreases. This accounts for the fact that notwithstanding good performance at the first stage, the *Sec* method is often the worst when these fragments are taken as the source from which fillers are extracted. Although *Lin* has the greatest positive/negative ratio, it performs poorly compared with other methods, because of inaccurate classifications at the initial stage. *Par* exhibited the best overall performance at the second level, outdoing *Sent* by a large margin.

4.2 Thresholding

We looked at how maximizing recall or precision interacts with different fragment sizes. We would like to see if thresholding can help to compensate for the weaknesses in a particular fragmentation method. Thus, we expect that overall performance of small fragments which greatly reduce the search space for C_2 can be improved by increasing recall for C_1. This will increase the search space for C_2, but the increase might be smaller than the one resulting from simply using larger fragments. Table 4 describes the results of these runs at C_2. In bold are the figures showing better performance than the runs without thresholding.

As will be noted from Table 3, boosting recall at C_1 does increase C_2's search space somewhat, but for the smaller fragments, *Sent* and *Lin*, the search space is still smaller than for *Sec* and *Par* without thresholding. As figures in Table 4 show, this leads to an

	Fragments			Tokens			Fragments			Tokens		
	P	R	F	P	R	F	P	R	F	P	R	F
	Sections						Sentences					
NAME	0.937	0.789	0.857	0.299	0.452	0.360	0.666	0.867	0.753	0.277	0.811	0.413
AREA	0.857	0.666	**0.750**	0.080	0.378	0.132	0.500	0.375	0.429	0.177	0.810	0.291
CREATOR	0.500	1	0.667	0.032	0.568	0.061	0.242	0.444	0.313	0.073	0.964	0.136
PLATFORM	0.466	1	0.636	0.291	0.388	0.333	0.818	0.692	0.750	0.288	1	0.447
PROGLANG	0.666	1	0.800	0	0	0	0.666	0.666	0.666	0.115	1	0.206
TGTLANG	0.400	1	**0.571**	0.233	0.304	0.264	0.103	0.800	0.183	0.041	0.761	0.078
EMAIL	0.304	1	**0.466**	0.169	0.907	0.285	0.134	1	0.236	0.173	0.975	0.294
AVERAGE	0.590	0.922	**0.720**	0.158	0.428	0.231	0.447	0.692	0.543	0.163	0.903	0.276
	Paragraphs						Lines					
NAME	0.924	0.910	0.917	0.525	0.792	**0.631**	0.875	0.913	**0.894**	0.277	0.480	0.351
AREA	0.363	0.500	0.421	0.301	0.814	**0.439**	0.600	0.375	0.462	0.245	0.285	0.263
CREATOR	0.476	0.625	0.540	0.319	1	**0.484**	0.636	1	**0.778**	0.029	0.509	0.055
PLATFORM	1	0.692	**0.818**	0.375	1	**0.545**	0.437	1	0.608	0.225	0.388	0.285
PROGLANG	1	1	**1**	0.636	0.700	**0.666**	0.400	1	0.571	0	0	0
TGTLANG	0.303	1	0.465	0.201	0.652	0.307	0.272	0.857	0.413	0.243	0.454	**0.317**
EMAIL	0.214	1	0.353	0.524	0.981	**0.683**	0.296	1	0.457	0.142	0.962	0.247
AVERAGE	0.611	0.818	0.848	0.700	0.412	**0.555**	0.502	0.878	0.639	0.166	0.440	0.241

Table 2: The accuracy of the fragment- and token-level classifiers resulting from each fragmentation method.

improvement in performance: both precision and recall rates frequently rose for *Sent*. In a similar fashion, larger fragments profit from increased precision. Boosting precision at C_1 narrows down the search space for C_2, which often improves the accuracy for larger fragments such as Sections.

4.3 Instance Selection

We examined instance selection techniques as an alternative way to relax the unbalanced data problem at C_2. We look at whether they are especially helpful for C_2 in situations when high recall at C_1 is achieved. The instance selection was carried out by randomly discarding negative instances from the training data until their count was the same as that of positive ones. Table 5 describes the results for different thresholding and fragmentation settings with instance selection applied. In bold are the results that are higher than those achieved using the same configuration but without instance selection.

We see that instance selection very often significantly improves recall, notably for *Sec* and *Lin*; the recall averages have gone up in all but one configurations. However, this is sometimes achieved at the cost of a considerable decrease in precision (e.g., for *Lin* from 0.114 to 0.038). This may be due to the fact that the model is induced from data that contains a greater proportion of positive instances. This causes the classification of a larger proportion of test instances as positive, hence higher recall, but lower precision. At the same time, in some cases, instance selection also resulted in an improved precision. In five out of the twelve configurations, the precision averages have increased. In general, it can be noted that instance selection helps to achieve greater effectiveness: for many configurations the averages of the F-measure have gone up. Cases, when the averages of the F-measure have deteriorated, are usually those when a large improvement in recall was achieved at the expense of very low precision. We believe that these unwanted situations can be avoided by finding a better proportion of positive and negative instances in the training data during instance selection.

4.4 Field-Specific Fine-Tuning

Using binary classifiers gives one the opportunity to adjust the classification problems for each template field separately. Table 6 compares the results achieved for each field using the most optimal configuration for that field (the last but one column) against the typical configuration of the double classification method, i.e. using sentence fragments without performing any thresholding or instance selection (the last column). The results indicate that fine-tuning the classification problem for each field separately offers a significant improvement over the traditional approach in terms of precision (by 0.2) and F-measure (by 0.26).

4.5 Identifying the Best Fillers

We evaluated the best filler identification algorithm against the performance of hand-crafted IE rules. The rules trigger the extraction of a particular field filler based on a variety of orthographic, linguistic, and page formatting cues. The hand-crafted rules were prepared by two domain experts; the construction of the rules took 4 person/weeks in total. As gold standard, we used the same evaluation data as in the previous experiments: a database was prepared by filling each template field for each document with the most frequent unique filler tagged by annotators in that document. The evaluation of the both IE methods consisted of 10-fold cross-validation, at each fold both methods were evaluated on the same set of documents.

We examined the effect of varying the parameters of the double classification method (the fragment size, thresholding and instance selection) on the perfor-

	Sec			Par			Sent			Lines		
	P	R	F	P	R	F	P	R	F	P	R	F
Boosted precision												
NAME	**0.305**	**0.456**	**0.366**	0.420	**0.828**	0.557	0.296	0.811	0.434	0.220	**0.481**	0.302
AREA	**0.085**	**0.400**	**0.140**	0.218	0.814	**0.344**	0.177	0.810	0.291	**0.274**	0.285	**0.279**
CREATOR	**0.018**	**0.551**	**0.035**	0.190	1	0.319	0.073	0.964	0.136	0.002	**0.142**	0.004
PLATFORM	**0.666**	0.200	0.308	0.230	1	0.374	0.288	1	0.447	**0.250**	0.272	0.261
PROGLANG	0	0	0	0.583	0.700	0.636	0.115	1	0.206	0	0	0
TGTLANG	**0.241**	**0.318**	**0.274**	0.141	0.652	0.232	0.041	0.761	0.078	0.243	0.454	0.317
EMAIL	0.142	**1**	0.249	0.358	0.981	0.525	0.173	0.975	0.294	0.063	**1**	0.119
AVERAGE	**0.208**	0.418	0.278	0.306	**0.854**	0.451	**0.166**	0.903	**0.280**	0.150	**0.376**	0.214
Boosted recall												
NAME	0.299	0.452	0.360	0.253	0.780	0.382	0.271	**0.900**	0.417	0.277	0.480	0.351
AREA	0.080	0.378	0.132	0.032	0.803	0.062	0.288	0.810	**0.425**	**0.041**	**0.213**	**0.069**
CREATOR	0.032	0.568	0.061	0.190	1	0.319	0.008	**1**	0.016	0.029	0.509	0.055
PLATFORM	0.291	0.388	0.333	0.230	1	0.374	0.428	0.923	**0.585**	0.225	0.388	0.285
PROGLANG	0	0	0	0.583	0.700	0.636	0.750	1	**0.857**	0	0	0
TGTLANG	0.233	0.304	0.264	0.096	0.652	0.167	0.003	**1**	0.006	**0.087**	0.391	**0.142**
EMAIL	0.169	0.907	0.285	0.358	0.981	0.525	0.041	**1**	0.079	0.142	0.962	0.247
AVERAGE	0.158	0.428	0.231	0.249	0.845	0.385	0.256	**0.948**	**0.403**	0.114	0.420	0.179

Table 4: The effect of boosting precision vs. recall at C_1 on the accuracy of C_2

.

mance of the best filler identification algorithm. Table 7 describes the results achieved with the most optimal parameter settings for each field (the last but one column) and compares them with the performance of the hand-crafted rules (the last column). We find that the performance of the proposed algorithm is consistently superior to that of the hand-crafted rules, and often by a considerable margin (e.g., by 0.83 for TGTLANG).

5 Conclusion

The double classification method provides convenient means to perform information extraction tasks where there is one template to be filled from an entire document. In this paper we presented an investigation into a number of parameters of the method in order to optimize its two classification subproblems and eventually improve its overall performance.

In general, these experiments have shown that finding appropriate settings for the three factors influencing the distribution of the task difficulty between the two classifiers helps to improve the performance of the method. In particular, doing so increased F-measure by 0.26 in comparison with using fragmentation of documents into sentences without applying thresholding and instance selection as was done in the original study by (Sitter & Daelemans 03).

The double classification method aims to extract all tokens instantiating of template fields, which is a very difficult and error-prone task. However, what is often needed instead is accurate extraction of one single filler which may consist of a single token or a sequence of tokens. We have presented a new method for the identification of such fillers in the output of the double classification method. The proposed method takes advantage of the evidence for the best filler in form of the relative position of tokens labeled as positive by the second classifier, the frequency of the token sequences, and the frequency of their subparts. Our evaluation shows that the method coupled with the double classification performs consistently better than hand-crafted extraction rules.

6 Acknowledgements

This study was conducted within the research project "An Automatic System for Resource Databases for Researchers" (ESRC grant RES-000-23-0010).

References

(Califf & Mooney 98) M. E. Califf and R. J. Mooney. Relational learning of pattern-match rules for information extraction. In *Working Notes of AAAI Spring Symposium on Applying Machine Learning to Discourse Processing*, pages 6–11, Menlo Park, CA, 1998. AAAI Press.

(Chieu & Ng 02) Hai L. Chieu and Hwee T. Ng. A maximum entropy approach to information extraction from semi-structured and free text. In *Proceedings of AAAI-02*, pages 768–791, 2002.

(Freitag 98) Dayne Freitag. Information extraction from html: application of a general learning approach. In *Proceedings of AAAI-98*, 1998.

(McCallum *et al.* 00) Andrew McCallum, Dayne Freitag, and Fernando Pereira. Maximum entropy Markov models for information extraction and segmentation. In *Proc. 17th International Conf. on Machine Learning*, pages 591–598. Morgan Kaufmann, San Francisco, CA, 2000.

(Mikheev 96) Andrei Mikheev. LT_CHUNK V 2.1. Language Technology Group, University of Edinburgh,UK, 1996.

(Sebastiani 02) Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1):1–47, 2002.

(Sitter & Daelemans 03) Ann De Sitter and Walter Daelemans. Information extraction via double classification. In *Proceedings of ATEM-2003*. Dubrovnik,Croatia, 2003.

(Soderland 99) Stephen Soderland. Learning information extraction rules for semi-structured and free text. *Machine Learning*, 34:233–272, 1999.

(Witten & Frank 99) Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 1999. ISBN: 1–558–60552–5.

	Sec			Par			Sent			Lines		
	P	R	F	P	R	F	P	R	F	P	R	F
no thresholding												
NAME	0.150	**0.857**	0.255	0.407	**0.867**	0.554	0.271	**0.900**	**0.417**	0.195	**0.880**	0.319
AREA	0.029	**0.864**	0.056	0.274	0.685	0.391	**0.288**	0.810	**0.425**	0.283	**0.775**	**0.415**
CREATOR	0.008	**1**	0.016	0.208	0.947	0.341	**0.097**	0.964	**0.176**	0.009	**1**	0.018
PLATFORM	0.005	**1**	0.010	**0.647**	0.916	0.758	**0.428**	0.923	**0.585**	0.006	**1**	0.012
PROGLANG	**0.714**	**0.500**	**0.588**	**1**	0.600	**0.750**	**0.750**	1	**0.857**	0.040	**1**	0.077
TGTLANG	0.003	**1**	0.006	0.009	**1**	0.018	0.016	**1**	0.031	0.004	**1**	0.008
EMAIL	0.008	**1**	0.016	0.050	**1**	0.095	0.041	**1**	0.079	0.008	**1**	0.016
AVERAGE	0.131	**0.889**	0.228	0.371	**0.859**	0.518	**0.270**	**0.942**	**0.420**	0.078	**0.951**	0.144
boosted precision												
NAME	0.162	**0.876**	0.273	**0.465**	**0.881**	**0.609**	**0.309**	**0.891**	**0.459**	0.159	**0.796**	0.265
AREA	0.033	**0.885**	0.064	**0.305**	0.666	**0.418**	**0.288**	0.810	**0.425**	**0.302**	**0.734**	**0.428**
CREATOR	0.006	**1**	0.012	**0.208**	0.947	**0.341**	**0.097**	0.964	**0.176**	**0.004**	**1**	**0.008**
PLATFORM	**0.800**	**0.400**	**0.533**	**0.647**	0.916	**0.758**	**0.428**	0.923	**0.585**	**0.636**	**0.636**	**0.636**
PROGLANG	**1**	**0.125**	**0.222**	**1**	0.600	**0.750**	**0.750**	1	**0.857**	**0.250**	**0.800**	**0.381**
TGTLANG	0.003	**1**	0.006	0.010	**1**	0.020	0.016	1	0.031	0.004	**1**	0.008
EMAIL	0.008	**1**	0.016	0.050	**1**	0.095	0.041	**1**	0.079	0.004	1	0.008
AVERAGE	**0.287**	**0.755**	**0.416**	**0.384**	**0.859**	0.531	**0.276**	**0.941**	**0.427**	**0.194**	**0.852**	**0.316**
boosted recall												
NAME	0.150	**0.857**	**0.255**	0.022	**1**	0.043	0.271	0.900	0.417	0.195	**0.880**	0.319
AREA	0.029	**0.864**	0.056	0.012	**1**	0.024	0.288	0.810	0.425	0.008	**1**	0.016
CREATOR	0.008	**1**	0.016	**0.208**	0.947	**0.341**	0.008	1	0.016	0.009	**1**	0.018
PLATFORM	0.005	**1**	0.010	**0.647**	0.916	**0.758**	0.428	0.923	0.585	0.006	**1**	0.012
PROGLANG	**0.714**	**0.500**	**0.588**	**1**	0.600	**0.750**	0.750	1	0.857	0.040	**1**	0.077
TGTLANG	0.003	**1**	**0.006**	0.006	**1**	**0.012**	0.003	1	0.006	0.003	**1**	0.006
EMAIL	0.008	**1**	**0.016**	0.050	**1**	**0.095**	0.041	1	0.079	0.008	**1**	0.016
AVERAGE	0.131	**0.889**	0.228	0.278	**0.923**	**0.427**	0.256	0.948	0.403	0.038	**0.983**	0.073

Table 5: The effect of instance selection on different fragmentation and thresholding configurations.

	Settings			Best settings			Typical		
	Thresholding	Fragmentation	Inst. sel.	P	R	F	P	R	F
NAME	boosted P	paragraph	yes	**0.47**	**0.88**	**0.61**	0.277	0.811	0.413
AREA	boosted P	paragraph	yes	**0.31**	0.67	**0.418**	0.177	0.81	0.291
CREATOR	boosted P	paragraph	yes	**0.21**	0.95	**0.341**	0.073	0.964	0.136
PLATFORM	boosted P	paragraph	yes	**0.65**	0.92	**0.758**	0.288	1	0.447
PROGLANG	boosted P or none	paragraph	yes	**1**	0.6	**0.75**	0.115	1	0.206
TGTLANG	boosted P	line	no	**0.24**	0.45	**0.317**	0.041	0.761	0.078
EMAIL	boosted P	paragraph	no	**0.36**	**0.98**	**0.524**	0.173	0.975	0.294
AVERAGE	-	-	-	**0.46**	0.78	**0.53**	0.163	0.903	0.276

Table 6: Comparison of accuracy using the best settings for each field against the typical parameter settings.

	Settings			One-best filler selection	Hand-crafted rules
	Thresholding	Fragmentation	Inst. sel.		
NAME	boosted P	paragraph	no	**0.527**	0.424
AREA	boosted P	sentence	yes	**0.705**	0.211
CREATOR	irrelevant	sentence	no	**0.639**	0.402
PLATFORM	irrelevant	sentence	yes	**1**	0.472
PROGLANG	boosted P or none	sentence	yes	**1**	0.443
TGTLANG	irrelevant	paragraph	no	**0.849**	0.016
EMAIL	irrelevant	paragraph	no	**0.276**	0.108
AVERAGE	-	-	-	**0.714**	0.129

Table 7: The F-measures of the best filler identification algorithm vs. hard-crafted rules.