Bootstrapping Information Extraction from Field Books

Sander Canisius and **Caroline Sporleder**
ILK / Communication and Information Sciences
Tilburg University, P.O. Box 90153, 5000 LE Tilburg, The Netherlands
{S.V.M.Canisius,C.Sporleder}@uvt.nl

Abstract

We present two machine learning approaches to information extraction from semi-structured documents that can be used if no annotated training data are available, but there does exist a database filled with information derived from the type of documents to be processed. One approach employs standard supervised learning for information extraction by artificially constructing labelled training data from the contents of the database. The second approach combines unsupervised Hidden Markov modelling with language models. Empirical evaluation of both systems suggests that it is possible to bootstrap a field segmenter from a database alone. The combination of Hidden Markov and language modelling was found to perform best at this task.

1 Introduction

Over the past decades much textual data has become available in electronic form. Many text types are inherently more or less structured, for example, classified advertisements for appartments, medical records, or logs of archaeological finds or zoological specimens collected during expeditions. Such documents consist of a number of shorter texts (or *entries*), each describing an individual object (e.g., an appartment, or an archaeological find) or event (e.g., a patient presenting to a health care provider). These descriptions in turn typically consist of different segments (or *fields*) which contain information of a specific type drawn from a more or less given inventory. Example (1), for instance, shows two descriptions of zoological specimens (a snake and three frogs) collected during an expedition. The descriptions contain different segments giving information about the specimens and the circumstances of their collection. For example, in the first description, *Leptophis* and *ahaetulla* refer, respectively, to the genus and species of the specimen, *road to Overtoom* mentions the place of collection, *in bush above water* encodes information about the biotope, *in the process of eating Hyla minuta* is a remark about the circumstances of collection, *16-V-1968* gives the collection date and *RMNH 15100* the registration number.

(1) Leptophis ahaetulla, road to Overtoom, in bush above water in the process of eating Hyla minuta 16-V-1968. RMNH 15100

Hyla minuta 1 ♀ 2 ♂ Las Claritas, 9-VI-1978 quaking near water 50 cm above water surface, near secondary vegetation, 200 m, M.S. Hoogmoed, RMNH 27217 27219

Unfortunately, this inherent structure is rarely made explicit. While the different object or event descriptions might be indicated by additional whitespace or other formatting means, as in the example above, the individual fields within a description are typically not marked in any way. However, knowledge of the inherent structure would be very beneficial for information extraction and retrieval. For instance, texts in their raw form only allow key word search. To retrieve all entries describing specimens of type *Hyla minuta* from a zoological field report, one can only search for occurrences of that string anywhere in the document. This can return false

Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, pp. 827–836, Prague, June 2007.

positives, such as the first description in (1) above, which does contain the string but is not about a *Hyla minuta* specimen but about a specimen of type *Leptophis ahaetulla* (the string *Hyla minuta* just happens to occur in the SPECIAL REMARKS field). On the other hand, if the genus and species information in an entry was explicitly marked, it would be possible to query specifically for entries whose GENUS is *Hyla* and whose SPECIES is *minuta*, thus avoiding the retrieval of entries in which this string occurs in another field.

The task of automatically finding and labelling segments in object or event descriptions has been referred to as *field segmentation* (Grenager et al., 2005).[1] It can be seen as a sequence labelling problem, where each text is viewed as a sequence of tokens and the aim is to assign each token a label indicating to what segment the token belongs (e.g., BIOTOPE or LOCATION). If training data in the form of texts annotated with segment information was readily available, the problem could be approached by training a sequence labeller in a supervised machine learning set-up. However, manually annotated data is rarely available. Creating it from scratch is not only time consuming but usually also requires a certain amount of expert knowledge. Moreover, the sequence labeller has to be re-trained for each new domain (e.g., natural history vs. archaeology) and possibly also each sub-domain (e.g., insects vs. mammals) due to the fact that the inventory of fields varies.

Thus, fully supervised machine learning is not feasible for this task. In this paper, we explore two approaches which require no or only a very small amount of manually labelled training data. Both approaches exploit the fact that there are often resources *derived* from the original documents that can potentially be utilised to bootstrap a sequence labeller in the absence of labelled training data. It is common practice, for example, that information contained in (semi-structured) field reports or medical records is manually entered into a database, usually in an attempt to make the data more accessible and easier to search. In such databases, each row corresponds to an entry in the original document (e.g., a zoological specimen) and the database columns correspond to the fields one would like to discern in the original document. Manually converting raw text documents into databases is a laborious task though, and it is rather common that the database covers only a small fraction of the objects described in the original texts. The research question we address in this paper is whether it is possible to bootstrap a domain-specific field segmentation system from an existing, manually created database for that domain. Such a system could then be applied to the remaining texts in that domain, which could then be segmented (semi-)automatically and possibly be added to the original database.

A database does not make perfect training material for a field segmenter though, as it is only derived from the original document and there are typically significant (and sometimes systematic) differences between the two data sources: First, while the ordering of the segments in a semi-structured text document is often not entirely fixed, some orderings are more likely than others. This information is lost in the derived databases. Second, the databases may contain information that is not normally present in the underlying text documents, for example information relating to the storage of an object in a collection. Conversely, some of the details present in the texts might be omitted from the database, e.g., the SPECIAL REMARKS field might be significantly shortened in the database. Third, pieces of information are frequently re-written when entered in the database, in some cases these differences may be systematic, e.g., dates, person names, or registration numbers might be written in a different format. Also, field boundaries in the text documents are sometimes indicated by punctuation, such as commas, and fields sometimes start with explicit key words, such as *collector*. Both of these features are missing from the database.

Despite of this, these databases will provide certain clues about the structure and content of different segments in the text documents. We exploit this in two different ways: (i) by concatenating database fields to artificially create annotated training data for a supervised machine learner, and (ii) by using the database to build language models for the field seg-

[1] The task differs from many other information extraction problems in which the aim is to extract short pieces of relevant information from larger text of largely irrelevant information. In *field segmentation*, all or most of the information in the input document is assumed to be relevant and the task is to segment it into fields containing different types of information.

mentation task.

2 Related Work

Most approaches to field segmentation and related information extraction tasks, such as filling templates with information about specific events, have been supervised. Freitag and Kushmerick (2000) combine a pattern learner with boosting to perform field segmentation in raw texts and in highly structured texts such as web pages and test this approach on a variety of field segmentation and template filling tasks. Kushmerick et al. (2001) address the problem of extracting contact information from business cards. They mainly focus on field *labelling*, bypassing the segmentation step by assuming that each line on a business card only contains one field (though a field like ADDRESS may span several lines). Their method combines a text classifier, for assigning likely labels to each field, with a trained Hidden Markov Model (HMM) for learning ordering constraints between fields. Borkar et al. (2001) identify fields in international postal addresses and bibliographic records by nesting HMMs: an outer HMM for modelling field transitions and a number of inner HMMs for modelling token transitions within fields. Viola and Narasimhand (2005) also deal with address segmentation but employ a trained context-free grammar.

One of the few unsupervised approaches is provided by Grenager et al. (2005), who perform field segmentation on bibliographic records and classified advertisements, using EM to fit an HMM to the data. They show that an unconstrained model does not learn the field structure very well and propose augmenting the model with a limited amount of domain-unspecific background knowledge, for example, by modifying the transition model to bias it towards recognising larger-scale patterns.

3 Learning Field Segmentation from Databases

3.1 Data

We tested our approach on two datasets provided by Naturalis, the Dutch National Museum of Natural History[2] Each dataset consists of (i) a number of field book entries describing the circumstances under which animal specimens were collected, and (ii) a database containing similar information about the same group of animals but in a more structured form. The latter were used for training, the former for testing. While the databases were manually created from the corresponding field books, we made sure that the field book entries we selected for testing did not overlap with the database entries. The two data sets are described below. Table 1 lists the main properties of the data.

Reptiles and Amphibians (RA) This dataset describes a number of reptile and amphibian specimens. The database consists of 16,670 entries and 41 columns. The columns relate, for example, to the circumstances of a specimen's collection, its taxonomic classification, how and where it is stored, who entered the entry into the database and when. Many database cells are empty. Those that are filled come in a variety of format, i.e., numbers, dates, individual words, and free text of various lengths. 22 of the columns contained information that was missing from the field books, e.g., information relating to the storage of the specimens; these columns were excluded from the experiments.

From the corresponding field books, 210 entries were selected randomly and manually annotated with segment information. To test the reliability of the manual annotation, 50 entries were labelled by two annotators. The inter-annotator accuracy on the token level was 92.84% and the kappa .92. The number of distinct field types found in the entries was 19, some of which only occurred in two entries, others occurred in virtually every entry. The average field length was four tokens, with a maximum average of 21 for the SPECIAL REMARKS field, and a minimum of one for fields such as SPECIES. The average number of tokens per entry was 60. Punctuation marks that did not clearly belong to any field were labelled as OTHER. In the experiments, 200 entries were used for testing and 10 for parameter tuning.

Pisces The second dataset contains information about the stations where fish specimens were caught. The database consists of 1,375 entries and four columns which provide information on the location of the stations. From the corresponding field books, we manually labelled 100 entries. Compared to the

[2] http://www.naturalis.nl

	RA	Pisces
# entries in DB	16,670	1,375
# fields	19	4
entry length (avg.)	60.17	39.79
segment length (avg.)	4.08	4.75

Table 1: Properties of the two datasets

first data set, this set is much more regular, with less variation in the number of segments per entry and in the average segment length. The field book entries are also much shorter and there are fewer segments (see Table 1).

3.2 Baselines

In order to get a sense of the difficulty of the task, we implemented five baseline approaches. For the first, **Majority** (**MajB**), we always assign the field label that occurs most frequently in the manually labelled test data, namely SPECIAL REMARKS. The other four baselines implement different look-up strategies, using the database to determine which label should be assigned to a token or token sequence.

Exact (**ExactB**) looks for substrings in a field book entry which exactly match the content of a database cell and then assigns each token in the matched string the corresponding column label from the database. There are normally several ways to match a field book entry to the database cells; we employed a greedy search, labelling the longest matching substrings first. All tokens that could not be matched in this way were assigned the label OTHER.

Unigram (**UniB**) assigns each token the column label of the database cell in which it occurs most frequently. If a token is not found in the database, it is assigned the label OTHER.

Trigram (**TriB**) assigns each token the most frequent column label of the trigram centred on it. If a trigram is not found in the database, the baseline backs off to the two bigrams covering the token and then to the unigram. If the token is not found in the database, OTHER is assigned.

Trigram+Voting (**TriB+Vote**) is based on a technique proposed by Van den Bosch and Daelemans (2005) for sequence labelling tasks. The main idea is to assign labels to trigrams in the sequence using a sliding window. Because each token, except the boundary tokens, is contained in three different trigrams (i.e., the one centred on the token to its left, the one centred on itself, and the one centred on the token to its right), each token gets three labels assigned to it, over which voting can be performed. In our case the labels are assigned by database look-up. If a trigram is not found in the database, no label is assigned to it. If the labels assigned to a given token differ, majority voting is used to resolve the conflict. If this does not break the tie (i.e., because all three trigrams assign different labels), the label of the trigram that occurs most frequently in the database is assigned. We also implemented two post-processing rules: (i) turning the label OTHER between two identical neighbouring labels into the surrounding labels, and (ii) labelling commas as OTHER if the neighbouring labels are not identical.

3.3 Supervised Learning from Automatically Generated Training Data

Our first strategy was to automatically generate training data for a supervised machine learner from the database. Since the rows in the database correspond to field book entries and the columns corresponds to the fields that we want to identify, training data can be obtained by concatenating the cells in each database row. The order of the fields in the field book entries is not fixed and this should also be reflected in the artificially generated training data. However, the field sequence is not entirely random, i.e., not all sequences are equally likely. If a small amount of manually annotated data is available, the field transition probabilities can be estimated from this, otherwise the best one can do is to assume uniform probabilities for all possible orderings. We experimented with both strategies, creating two different training sets, one in which the database cells were concatenated randomly with uniform probabilities, and another in which the cells were concatenated to reflect the field ordering probabilities estimated from ten entries in the manually labelled development set.[3] When estimating the field transition

[3]We found that 10 annotated entries are enough for this purpose; the field segmentation results we obtained by estimating the sequence probabilities for the training set from 100 entries were not significantly different. This is probably because the probabilities are only used indirectly, i.e. to bias the field orderings for the generated training data. If the probabilities were used directly in the model, the amount of manually annotated data would probably matter much more.

probabilities, we computed a probability distribution over the initial fields of an entry as well as the conditional probability distributions of a field x following a field y for all seen segment pairs in the ten entries. To account for unobserved events, we used Laplace smoothing.

The artificially created training data were then converted to a token-based representation in which each token corresponds to an instance to be labelled with the field to which it belongs. On the whole, we had just under 700,000 instances (i.e., tokens) in our training data. We implemented 107 features, falling in three classes:

- the neighbouring tokens (in a window of 5 centering on the token in focus)
- the typographic properties of the focus token (word vs. number, capitalisation, number of characters in the token etc.)
- the tfidf weight of the focus token in its context with respect to each of the columns in the database (i.e., the fields)

The tfidf-based features were computed for a window of three, centering on the token in focus. For all n-grams in this window covering the token in focus (i.e., the trigram, the two bigrams, and the unigram of the focus token), we calculated the $tfidf$ similarity with the columns in the database, where the similarity between an n-gram t_i and a column col_x is defined as:

$$tfidf_{t_i,col_x} = tf_{t_i,col_x} \log idf_{t_i}$$

The term frequency, tf_{t_i,col_x} is the number of occurrences of t_i in col_x divided by the number of occurrences of all n-grams of length n in col_x (0 if the n-gram does not occur in the column). The inverse document frequency, idf_{t_i}, is the number of all columns in the database divided by the number of columns containing t_i. A high tfidf weight for a given n-gram in a given column means that it frequently occurs in that column but rarely in other columns, thus it is a good indicator for that column.

The training data was then used to train a memory-based machine learner (TiMBL (Daelemans et al., 2004), default settings, $k = 3$, numeric features declared) to determine which field each token belongs to.[4]

3.4 Hidden Markov Models

Our second approach combines language modelling and Hidden Markov Models (HMMs) (Rabiner, 1989). Hidden Markov Models have been in use for information extraction tasks for a long time. A probabilistic model is trained to assign a label, or *state* to each of a sequence of observations, where both labels and observations are expected to be sequentially correlated; hence the popularity of HMMs in natural language processing and information extraction. Recently, a large number of more sophisticated learning techniques have largely replaced HMMs for information extraction; however unlike most of those newer techniques, HMMs offer the advantage of having a well-established unsupervised training procedure: the Baum-Welch algorithm (Baum et al., 1970).

Training a Hidden Markov Model, whether supervised or unsupervised, comes down to estimating three probability distributions.

1. An initial state distribution π, which models the probability of the first observation of a sequence to have a certain label.
2. A state-transition distribution A, modelling the conditional probability of being in a certain state s, given that the previous state was s'.
3. A state-emission distribution B, which models the conditional probability of observing a certain object o given some state s.

For information extraction tasks, the typical interpretation of an *observation* as referred to above, is that of a token, where the entire observation sequence commonly corresponds to one sentence. In the current study, we chose to apply HMMs on a somewhat higher level, where an observation corresponds to a *segment* of the field book entry. Ideally, one such segment maps one-to-one to a cell in the specimen database, though we leave open the possibility of merging several segments into one database cell.

Provided that a field book entry can be segmented reliably, we have turned one part of the learning problem, that of estimating the state-emission

[4] We chose TiMBL because it has been applied successfully to sequence labelling tasks (Van den Bosch and Canisius, 2006; Van den Bosch and Daelemans, 2005).

distribution, into one for which we have (almost) perfect supervised training data: the contents of the database cells. The general form of a Hidden Markov Model's state-emission distribution is $P(o|s)$, where s is the state, i.e. a field type in our case, and o is the observation. As mentioned before, we treat a segment of tokens as one observation, therefore our state-emission distribution will look like $P(o = t_1, t_2, ..., t_n|s)$. Essentially, what we have here is a language model, conditioned on the current state. Since the specimen database provides a large amount of labelled segment sequences, any probabilistic language modelling method can be used to estimate the state-emission distribution.

Whereas the specimen database provides sufficient information to estimate the state-emission distribution in a fully supervised way, the initial-state and state-transition distributions cannot be derived from the database alone. Columns in a database are either unordered or ordered in a way that does not necessarily reflect the order they had in the field book entries they were extracted from. However, the original field book entries do show a rather systematic structure. Often, using information about the order fields typically occur in, seems to be the only way to distinguish certain field types from one another. To estimate the two missing probability distributions, the Baum-Welch algorithm was used, updating the initial-state and state-transition distributions, while keeping the state-emission distributions unchanged.

3.4.1 Segmentation of Field Book Entries

In our setup, the Hidden Markov Model expects the input texts to be pre-segmented. To come up with a good initial segmentation of an input entry, we again chose a language-modelling approach. It is expected that segment boundaries can best be recognised by looking for unusual token subsequences; that is, token sequences that are highly unlikely to occur within a field according to the information we obtained from the specimen database about what a typical segment does look like. A bigram language model has been trained on the contents of *all* the columns of the specimen database. Using this language model and the Viterbi algorithm, the globally most-likely segmentation of the input text is predicted.

3.4.2 The State-emission Model

The state-emission model is constructed by training a separate bigram language model for each column of the specimen database. Combining those gives us the conditional distribution required for a Hidden Markov Model. However, in a database, not every column has necessarily been filled for every record. For example, in the Reptiles and Amphibians database, there are columns that only contain actual data as infrequently as in 5% of the records. Relative to columns that contain data more often, these sparsely-filled columns tend to be overestimated when simply computing a likelihood according to the language model. For this reason, a penalty term is added to the state-emission distribution corresponding to the probability that a record contains data for the given column. The likelihood computed by the language model and the corresponding penalty term are then simply multiplied.

3.4.3 Language Modelling

For building both types of language model presented in the two previous sections, we used n-gram language modelling as implemented by the SRI Language Modelling Toolkit (Stolcke, 2002). With this toolkit, high-order n-gram models can be built, where the sparsity problem often encountered with such models is tackled by various smoothing methods. We supplemented this built-in n-gram smoothing, with our own smoothing on the token level by replacing low-frequent words with symbols reflecting certain orthographic features of the original word, and numbers with a symbol only encoding the number of digits in the original number.

In addition to these general measures to deal with sparsity, we also applied a small number of knowledge-driven modifications to the training data for the language models. The need for those is caused by the fact that the contents of the specimen database are almost, but not entirely extracted literally from the original field book entries. For example, for the second entry of Example 1, the following information is stored in the database.

Genus Hyla

Species minuta

Gender 1 f + 2 m

Place Las Claritas

Collection date 9-6-1978

Biotope quaking near water 50 cm above water surface, near secondary vegetation

Height 200 m

Collector M.S. Hoogmoed

Registration number 27217 27219

Comparing just this single field book entry with its corresponding database record, one can already see several mismatches. The gender symbols ♂ and ♀ in the field book texts are stored as `m` and `f` in the database. The collection date 9-VI-1978, has been converted to 9-6-1978 before adding it to the database, i.e. the Roman numeral for the month number has been mapped to the corresponding Arabic numeral. As a final example, in the field book entry the registration number for the specimen is preceded by the symbol `RMNH`; in the database this standard symbol is stripped of and only the number is stored. Each of these differences, while only small, will hinder the performance of a language model trained on the contents of the database and applied to field book texts.

As a simple illustration of this, when encountering the symbol `RMNH` in a field book entry, this most likely indicates the start of a new (registration number) segment. However, in the database, on which all language models are trained, `RMNH` never occurs as a symbol in the registration number column; it does occur a few times in the column for special remarks but never at the start of the text. As a result, a segmentation model trained on the contents of the database, on encountering the symbol `RMNH` will always opt for continuing the existing segment as opposed to starting a new one, which is most likely the better choice.

Fortunately, many such mismatches between the text in field books and the database are systematic and can easily be covered by a small number of manually constructed rules that modify the training data for the language models. Among others, we added the `RMNH` symbol in front of registration numbers, and randomly changed some month numbers from Arabic numerals to Roman numerals.

Another difference between the field books and the database that turned out to be rather crucial is the fact that many segments in the field book entries are separated by commas. Such commas used as delimiters *between fields* do not appear in the database, where fields correspond to columns and boundaries between fields consequently do not have to be explicitly marked by punctuation. For example, the comma between Las Claritas and 9-VI-1978 only serves to separate the *Place* segment from the *Collection date* segment; the comma is not copied to the database. However, commas do occur *field-internally* in the database, especially in longer fields such as SPECIAL REMARKS. Hence a language model trained on the database in its original form will never have encountered a comma functioning as a segment boundary marker and thus will not recognise that commas may be used for this purpose in the field book entries. To deal with this, we modified the training data for the segment model by randomly inserting commas at the end of some segments. Experimental results point out that this modification has a large impact on the performance of the segmentation model.

	Token	Segment			
	Acc.	Prec.	Rec.	$F_{\beta=1}$	WDiff
MajB	24.8	0.0	0.0	0.0	.346
ExactB	16.0	25.7	23.1	24.3	.425
UniB	27.0	8.9	22.8	12.8	.818
TriB	43.8	12.9	24.8	16.9	.582
TriB+Vote	45.1	14.9	27.8	19.4	.536
MBL rand.	44.6	7.1	19.2	10.4	.568
MBL bias	53.4	12.1	32.0	17.6	.533
HMM	56.9	62.7	58.1	60.3	.177

Table 2: Performance of all baseline and learning approaches on the Reptiles and Amphibians data, expressed in token accuracy, precision, recall, F-score, and WindowDiff. For WindowDiff, lower scores are better.

3.5 Results and Discussion

To evaluate the performance of the two approaches, we applied them to the Reptiles and Amphibians database. First we computed baseline scores using the approaches described in Section 3.2. All resulting scores are listed in Table 2.

Performance of the systems was measured using a number of different metrics, each reflecting different qualities of the output. The most basic one, token accuracy, simply measures the percentage of tokens that were assigned the correct field type. It has the disadvantage that it does not reflect the quality of the segments that were found. For a more segment-oriented evaluation, we used precision, recall and F-score on correctly identified and labelled segments. As a last measure for segmentation quality we used WindowDiff (Pevzner and Hearst, 2002), which only evaluates segment boundaries not the labels assigned to them. In comparison with F-score, it is more forgiving with respect to segment boundaries that are slightly off.

The baseline performance scores support our assumption that the contents of the database can be used to learn how to segment and label field book entries, i.e. the increasingly more sophisticated database matching strategies each cause a substantial performance improvement up to 45.1 token accuracy for the trigram lookup with voting strategy. The biggest problem of all baseline approaches is that their performance with respect to the segment-oriented measures is disappointing. Even trigram lookup with voting only reaches an F-score of 19.4.

Looking at the performance of the two memory-based learners in Table 2 (*MBL rand.* was trained on randomly concatenated training data, *MBL bias* on data modelled after 10 training sequences), we see that the small amount of prior knowledge used for generating the artificial training data results in a substantial improvement compared with the memory-based learner that was trained on randomly concatenated training data with uniform probabilities.

As can be seen in the last row of Table 2, the Hidden Markov Model outperforms all other approaches in all aspects; it attains both the best token accuracy (56.9), and by far the best F-score (60.3). The most probable explanation for the superior performance of the HMM-based approach is that this approach models sequential constraints between different segments, whereas the baselines and the memory-based models are predominantly local.

In Table 3, we consider the effect that the knowledge-based rewriting rules discussed in Subsection 3.4.3 have on the performance of both the segmentation and the labelling step. We evaluate both (i) the performance of the two processing steps separately—for labelling this presupposes perfectly segmented input— and (ii) the performance of the cascade of segmentation and labelling. As before, the performance of the labelling and the cascade is expressed in F-score on segments. Performance of the segmentation is measured in F-score on inserted segment *boundaries*.

The first row of the table shows the scores if no modification rules are used. This proves detrimental for the segmentation, only attaining an F-score of 28.4. With 62.3, the F-score for labelling is reasonable; however, the weakness of the segmentation causes the output of the entire cascade to be useless. Modifying the training data for the segmentation model by randomly inserting a comma at the end of segments gives a substantial improvement in segmentation performance, and as a result the quality of the cascade improves with it, as can be seen in the second row. All remaining rows list the scores of a single modification rule applied to the training data *in addition to* the comma rule. Each of the rules gives a slight performance increase. Using all rules together makes a big difference: the F-score of the cascade increases from 44.7 with the comma rule only, to 60.3.

Rule	Boundaries	Labels	Cascade
None	28.4	62.3	17.2
Comma	69.7	62.3	44.7
Comma+Collection Date	69.7	64.4	46.8
Comma+Reg. Number	72.9	68.6	50.9
Comma+Gender	71.5	65.0	49.5
Comma+Collector	70.4	65.8	45.3
All	72.0	78.3	60.3

Table 3: The effect of systematically modifying the training data for both the segmentation and labelling models. The comma rule is used only in the training of the segmentation model. The other rules are named after the database field they are applied to. The scores reflect their performance when applied in conjunction with the comma rule.

To confirm that the HMM-based approach carries over to other datasets, we also tested it on the Pisces data. The results of this experiment, as well as all baseline scores are presented in Table 4.[5] The fact that this data set is more regular and contains fewer segments is reflected by the relatively high token accuracies attained by the baseline approaches. With

[5]We did not test the memory-based approaches as these led to significantly worse results than the HMM-based model on the Reptiles and Amphibians data set.

the simple database lookup strategies, however, almost no entire segments are predicted correctly. Attaining a token accuracy of 94.4 and an F-score of 86.9, the performance of the Hidden Markov Model is again more than satisfactory, confirming the results observed with the Reptiles and Amphibians data.

	Token	Segment			
	Acc.	Prec.	Rec.	$F_{\beta=1}$	WDiff
MajB	50.0	0.0	0.0	0.0	.279
ExactB	9.3	0.0	0.0	0.0	.565
UniB	53.7	1.5	3.6	2.1	.588
TriB	68.6	0.2	0.2	0.2	.359
TriB+Vote	67.1	2.2	2.8	2.4	.384
HMM	94.4	87.6	86.3	86.9	.049

Table 4: Performance of Hidden Markov Model and all baseline approaches on the Pisces data, expressed in token accuracy, precision, recall, F-score, and WindowDiff. For WindowDiff, lower scores are better.

4 Conclusion

Information extraction is often used to automate the process of filling a structured database with content extracted from written texts. Supervised machine learning approaches have been successfully applied for creating systems capable of performing this task. However, the supervised nature of these approaches requires large amounts of annotated training data; the acquisition of which is often a laborious and time-consuming process. In this study, we experimented with two machine learning techniques that do not require such annotated training data, but can be trained on a database containing information derived from the type of documents targeted by the application.

The first approach is an attempt to employ a standard supervised machine learning algorithm, training it on artificial labelled training data. These data are created by concatenating the contents of the cells of the database records in random order. Experiments with this approach pointed out that truly random concatenation of database fields results in weak performance; a rather simple baseline approach, which only matches substrings of a field book entry with the contents of the database, leads to better results. However, if a small amount of annotated field book entries is available —in this study, 10 entries turned out to be sufficient— one can estimate field ordering probabilities that can be used to generate more realistic training data from the database. A machine learner trained on these data labelled 10% more tokens correctly than the system trained on the randomly generated data.

Our second approach is based on unsupervised Hidden Markov modelling. First, an n-gram language model is used to divide the field book entries into unlabelled segments. Then, a Hidden Markov Model is trained on these segmented entries using the Baum-Welch algorithm to estimate state-transition probabilities. The resulting HMM labels the segments found in the preceding segmentation step. The HMM state-emission distributions are estimated by training n-gram language models on the contents of the database columns.

The performance of the HMM proved to be superior to the other approaches, outperforming the supervised learner by labelling 56.9% of the tokens correctly, as well as attaining good results in terms of segment-level F-score (60.3). Experiments with the HMM approach on a second, independent data set confirmed its generality.

Acknowledgements

The research reported in this paper was funded by the Netherlands Organisation for Scientific Research (NWO) as part of the IMIX and CATCH programmes. We are grateful to Antal van den Bosch, Marieke van Erp, Steve Hunt, and the staff at Naturalis, the Dutch National Museum for Natural History, for interesting discussions and help in preparing the data.

References

Leonard E. Baum, Ted Petrie, George Soules, and Norman Weiss. 1970. A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains. *The Annals of Mathematical Statistics*, 41(1):164–171.

Vinayak Borkar, Kaustubh Deshmukh, and Sunita Sarawagi. 2001. Automatic segmentation of text into structured records. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, pages 175–186.

Walter Daelemans, Jakub Zavrel, Ko Van der Sloot, and Antal Van den Bosch, 2004. *TiMBL: Tilburg Memory Based Learner, version 5.1, Reference Guide*. ILK Research Group Technical Report Series no. 04-02.

Dayne Freitag and Nicholas Kushmerick. 2000. Boosted wrapper induction. In *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI/IAAI-2000)*, pages 577–583.

Trond Grenager, Dan Klein, and Christopher D. Manning. 2005. Unsupervised learning of field segmentation models for information extraction. In *Proceedings of the 43nd Annual Meeting of the Association for Computational Linguistics (ACL 2005)*, pages 371–378.

Nicholas Kushmerick, Edward Johnston, and Stephen McGuinness. 2001. Information extraction by text classification. In *Proceedings of the IJCAI-01 Workshop on Adaptive Text Extraction and Mining*.

Lev Pevzner and Marti A. Hearst. 2002. A critique and improvement of an evaluation metric for text segmentation. *Computational Linguistics*, 28(1):19–36.

Lawrence R. Rabiner. 1989. A tutorial on Hidden Markov Models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286.

Andreas Stolcke. 2002. SRILM - an extensible language modeling toolkit. *Proceedings of the International Conference on Spoken Language Processing (ICSLP 2002)*, 2:901–904.

Antal Van den Bosch and Sander Canisius. 2006. Improved morpho-phonological sequence processing with constraint satisfaction inference. In *Proceedings of the Eighth Meeting of the ACL Special Interest Group in Computational Phonology (SIGPHON '06)*.

Antal Van den Bosch and Walter Daelemans. 2005. Improving sequence segmentation learning by predicting trigrams. In *Proceedings of the Ninth Conference on Natural Language Learning, CoNLL-2005*, pages 80–87.

Paul Viola and Mukund Narasimhand. 2005. Learning to extract information from semi-structured text using a discriminative context free grammar. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 330–337.