

User Driven Example-Based Training for Creating Lexical Knowledgebases

Jon D. Patrick, Dusan Palko, Robert Munro, Michele Zappavigna

Sydney Language Technology Research Group
School of Information Technologies,
University of Sydney, NSW, Australia

{jonpat, dpalko, rmunro, michele}@it.usyd.edu.au

Abstract

The task of creating a lexical knowledgebase has been defined in our work as extracting appropriate semantic phenomena from what are essentially print dictionaries stored in a desktop publishing format. The aim of this work is to achieve automatic identification of structural elements in the dictionary's stream of text that are isomorphic to semantic fields of interest. A fully automatic solution appears infeasible to this problem. A semi-automatic approach that exploits training from a knowledgeable user to identify structures is shown to be fruitful. Once learnt from the user the structures are then applied automatically to other text streams in the same document or to other documents. On completion of training the data records are written into a lexical knowledgebase in an XML mark up format. Experimental work demonstrates the effectiveness of the system by a user training the system on a minimal set of entries in a dictionary. The usefulness of the learned rules is demonstrated with a 4 fold increase in processing throughput after a training of only 178 records in 138 minutes. Key features of the software system, Ferret III, are ease of use, fast performance and robustness of the system.

1 Introduction

A study of the literature shows little attention to finding methods to support rapid inference of structure in dictionaries although many general inferencing strategies from machine Learning have been in NLP. However, there are discussions on the issues of converting dictionaries into machine readable forms and their use for computational tasks. The work of Ide & Veronis (1995) deals with the problem of creating the TEI standard (Sperberg-McQueen & Burnard, 1994) for print dictionaries and some discussion of the competing needs to satisfy both the different users of dictionaries, readers versus computational linguists, and the level of generality of description, that is the ability to describe the differing structures between dictionaries. Tutin & Veronis (2001) further identify that the TEI standard for the SGML markup of dictionaries creates limitations, especially for the editorial or presentational view, and is better to be adapted for particular needs with the introduction of new elements but such adaptations need to be managed in a strict manner to allow direct translation back to the original standard.

Ide & Veronis make the point that dictionaries are arguably the most complicated of all document types. The surface structure of a dictionary is highly variable yet they retain a deeper semantic structure that users exploit effectively. Much of the deeper structure is not explicit in the surface, but requires knowledge of the dictionary conventions for both abbreviations and layout. They make the further comment that duality of surface structure and deep knowledge in dictionaries leads to the two different views, that is the textual view of the presentation needs versus the "database" view needed for computational processing. In fact they

assert that “the kind of inferencing required to retrieve deep structure information from the surface structure may be difficult if not impossible, for a computer to accomplish” (p168). We concur with this view as a generalization but address the issue of semi-automatic machine learning of the deeper structure through user assisted training of the rules of entry decomposition.

The task of creating a lexical knowledgebase has been defined in our work as extracting appropriate semantic phenomena from what are essentially print dictionaries stored in a desktop publishing format. The term “knowledgebase” has been used by Turcato, Popowich, McFetridge & Toole (2000) in a restricted sense to represent only stores of collections of Lexical Transfer Rules (LTRs), that is rules about the correspondence of lexical items across languages for word and phrase equivalencies, morphological and syntactic information, and functional dependencies. Like Tourcato et al our work is with multi-lingual dictionaries however they contain more extensive content than encompassed in their definition although not always in the quite explicit formalism used in their computing system. Such extra data are ontological classes and semantic context. Hence we use the term ‘knowledgebase’ with a wider scope than Turcato et al.

2 Semi-Automated Learning of Text Structure

Automated learning and discovery by inductive inference of the inherent meaningful structure of an arbitrary stream of text is a difficult task, both on a theoretical level and from the perspective of designing and implementing a workable software solution. The general solution will inevitably be complex because it not only involves an element of statistical modeling but also has to achieve the requirements of flexibility, ability to handle a large volume of data and processing throughput.

Our aim is to develop a method for automated learning of semantic fields from inherent structure in a collection of text records from a dictionary and convert them into a form suitable for use as a computational lexical knowledgebase. As achieving fully automated mapping from text layout to semantic mark-up appears intractable we have

opted for a semi-automatic approach by exploiting the knowledge of a user to train a machine learner to perform the transformation task. A software solution needs to provide a user with full functionality to read a data source file and specify an inherent structure present in the data. It should be able to record and save the systematic rules derived from user actions as they prime the data identifying its structure and be able to reapply them automatically to successive records in the data file. The rules should also be part of a mechanism of easy and straightforward application of a powerful statistical apparatus for further automated inductive inference of structure. The case study on which our ideas are tested is an English-Basque multi-lingual dictionary stored in an RTF format.

3 Architectural Design of a User-Driven Training System

The processing problem requires an approach from Machine Learning where it is necessary to learn the structure of the data by analysing features in the data stream. We have previously attempted to solve this problem from a bottom-up perspective where the data stream was considered to be a stream of characters and the learners would assemble character by character compound terms (RTF tags) which in turn are subjected to repeated inferences of more compound terms. This strategy was first presented by Solomonoff (1964) in a general inductive inference context. However in our case, this approach failed due to the very large variety of combinations of mark-up tags and a lack of user input to guide the direction of training.

Our second solution, described here, can be considered a top-down approach, where the user controls the induction process by indicating boundaries for subdividing the data records along desirable semantically meaningful units. Such an approach exploits high quality human knowledge about the structure of the data stream. However, we wish to overcome the labour intensive process of marking the structure of every single entry in a large document, for example the 22,000 entries in our dictionary. Hence we would rather reuse the rules defined for one entry on all other entries in the document. This approach requires three structural elements: a method for storing the rules described

by the user, a method for reapplying a rule defined for one entry on other entries, and a mechanism for allowing the user to express the rules or structure of an entry. The solution to the first and second elements has been to devise a single strategy whereas the third element is an interface design problem ultimately resolved by our own intuitions about the easiest way to perform the functions. A further stage is required to convert the decomposed entry into a form suitable for computational use as a knowledgebase. This is achieved by designing a lexical knowledgebase schema and describing it in an XML DTD and then automatically writing the decomposed entry into an XML database management system for storage.

The solution to storing and reapplying the structural rules of the entries is achieved by representing them as transitions in a Finite State Automaton (FSA). An enhancement on this solution is to extend the FSA to a Probabilistic FSA (PFSA) and so capture the frequency at which different rules are exploited throughout the full set of entries in the data stream. This information can be used in turn for a number of other functions, such as inferring a minimal structure for the PFSA, and identifying rarely used structures that suggest errors in either the dictionary organisation or rule annotation.

The process of recording rules is primarily based on identifying tokenised RTF tags in the text as transitions from one semantic field to another and to label the nodes of the PFSA with names of the semantic fields. The current system is entirely restricted to using RTF tags for identification. It is desirable to exploit surrounding text in boundary identification where the RTF tag is not alone sufficient. For example punctuation is often a boundary marker alone. However, exploitation of local text significantly lengthens the training time and the size of the learnt automata. The data, that is the text stream, at no time is stored in the PFSA, however it may be modified in memory to assist in resolving parsing ambiguities if other available methods are insufficient.

4 Implementation Outline

As the software is expected to deal with significant amounts of data, that is tens of thousands of dictionary entries, there is a need to save as much

memory and CPU resources as possible in order to provide a reasonable balance between performance and user-friendliness. By “reasonable” balance we mean that the user is provided with an easy to use graphical environment yet not be distracted by sluggish performance while the application is performing simple operations. Earlier prototypes without a separation of the PFSA from the data records were found to be particularly slow at presenting the analysed text on the screen as one moved from record to record.

The system consists of 4 modules. The tokeniser module converts the initial data from the RTF file format to a standardized and in some way structured form. It transforms the source file into a collection of homogeneous data items. This collection is then passed to the PFSA module which is the core part of the application and contains a set of methods which provide functionality for priming and learning. The third module is the GUI which provides the mechanisms for the user to insert rules and view the semantic segmentation of the data. The fourth module accepts a template for the schema description of the lexical knowledgebase in the form of a set of XML marked up examples, converts the FSA representation of the dictionary structure into the XML template and writes the data records into an XML database.

5 Tokeniser Module

The tokeniser module must be able to process the format of the source data. In our experiments so far we have only processed RTF files. We are currently modifying the system to allow specification of other file formats by parameter files.

Conceptually, the tokeniser is a pre-processor that converts the data source file into a collection of tokens. There are only two types of tokens, namely collections of (RTF) mark-up tags and data string tokens. The RTF tags need to be tokenised because they themselves are made up of a variety of tags separated by a ‘\’ indicating many different elements for formatting and typesetting. Non-RTF information that might be used for segmentation such as punctuation, is not handled by the system. It is a problem of significant scale to exploit non RTF tag information and we are currently endeavouring to identify a general solution.

6 PFSA Module

The PFSA is designed to be independent of the data stream, that is, no data is actually stored in the PFSA. It only stores the name states and transition token sets, which represents the user-desired segmentations demarcating the boundaries of relevant semantic fields. One perspective to understand this arrangement is to see that the data exists as entries in a file and the PFSA as a free-standing lattice that can be overlaid on the entries to show their segmentation. This overlaying only occurs when the data is displayed through the user interface. On any request for display of entries, the PFSA scans an entry and segments it on the display window according to the tokens in the entry that match transition tokens in the PFSA. The data is never physically segmented in the computational process.

Hence, in our design the system does not construct structures that store data pieces belonging to specific states. Instead, it utilizes the initial data (in suitably transformed form). This design turns out to be extremely efficient, since it avoids unnecessary data duplication, and results in high per-

formance. Effectively, all data segmentation are performed “on the fly”, which shifts the main performance aspect into CPU capability rather than storage/retrieval operations, which are significantly slower by their nature.

This separation creates a number of other advantages. Any PFSA can be reused at any time with any other data set. Multiple PFSA's can be readily constructed for different data subsets. Storage of a PFSA and data set are independent allowing other processing of the data.

7 Graphical User Interface (GUI) Module

FERRET is a window-based application providing a typical graphical interface that can be easily operated by a mouse. It contains a main frame with menu bar attached to the top edge of the frame. The display is divided into two separate windows by a horizontal line. The top window displays the full text of a record and the bottom window displays the current record divided into the fields for which it has been primed. To browse the records, the user can use either the menu, or two large but-

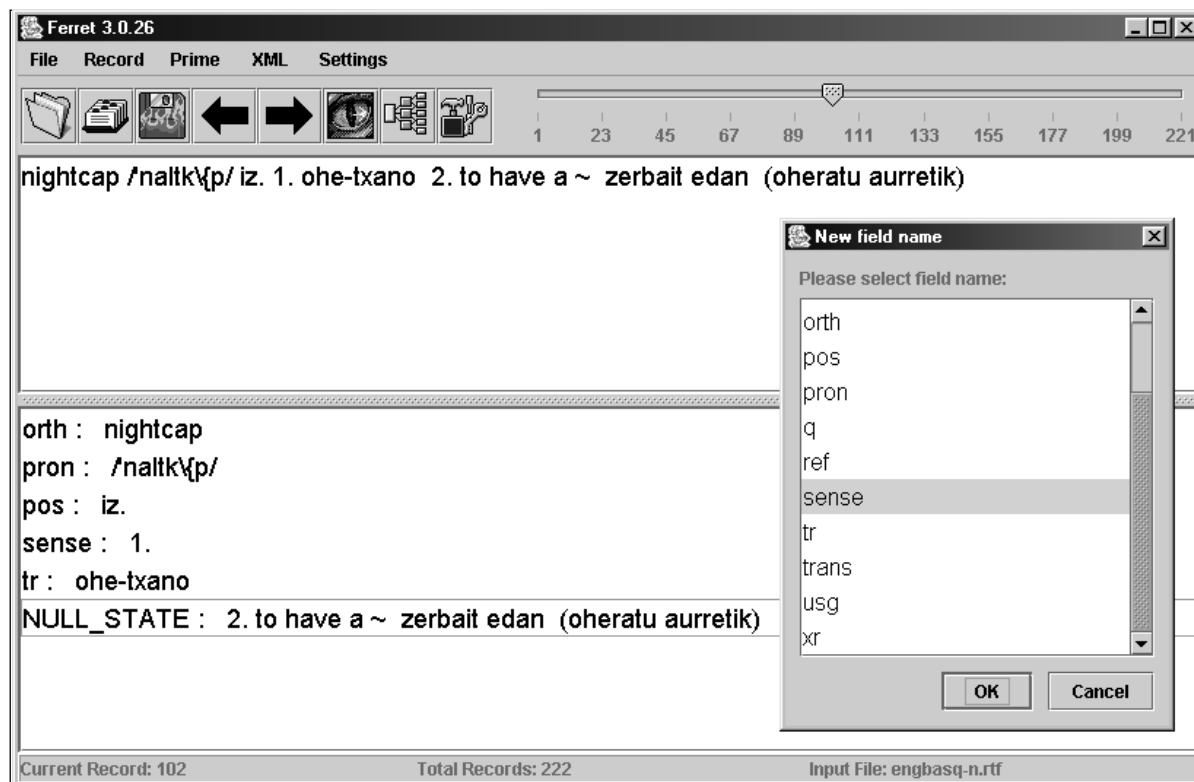


Figure 1. The GUI interface of Ferret during a priming operation.

tons marked as “Next Record>>” and “<< Previous Record”. At the bottom of the screen is a status bar displaying current record number and total number of records read from the RTF file. A menu option exists to output the data file as an XML marked up knowledgebase. Figure 1 shows the FERRET GUI whilst in the operation of priming. The small pane on the right is the window in which the user can see the text segmented according to the RTF tags embedded in the text. The entry has been trained for the first sense and is awaiting training of the second sense (contents of the NULL_STATE line).

8 Inferring User Rules through Priming

The system learns through the priming function being executed on a data file of records by the user. The priming function allows the user to view entries and systematically divide an entry into semantically meaningful components. Each prime performed by the user is immediately available to all records in the file in memory. Each time the user views another record then all priming information currently held in the PFSA is used to display the data records semantic partitioning.

In the case of dictionaries, the typical components are headword, pronunciation, meaning, example, reference, and others, depending on the context. The process of priming involves the interface presenting an entry subdivided according to the RTF tags that the tokeniser has been able to

identify. The user then indicates which segment boundaries are meaningful to them and labels the field with an appropriate name. The tag or tag set is treated as a transition symbol for the PFSA and added to its description in the appropriate place in the PFSA. In this way rules of the user are captured as a priming process but are independent of any particular entry as the entry is unmodified. At times the user’s semantic field involves concatenating a number of data strings separated by RTF tags. In this case the learner is able to capture the sequence of tags that are necessary for the transition through the data string set and to re-use that sequence on any other record.

9 Generating the Lexical knowledgebase

The XML output generation is divided into two stages: inferring the desired XML structure and converting the PFSA and data records to a file in this format.

The user must supply Ferret III with an example of a knowledgebase marked up in XML. We have prepared this file with 100 examples drawn from our dictionaries to represent the variety of structures that we expect to use. The inference engine uses the example file and derives a grammar describing legal nesting XML structures. In essence, it infers a DTD. It is stored as a directed graph, with a child node representing a tag set that may legally be nested immediately within the tag set represented by the parent node. Re-

File	No. Records	No Fields	No. Primes/ Markup	Time to Process (min)	Primes/ Rec	Fields/ Prime	Fields/ Rec	Fields/ min
1- letter X	22	128	128	16	3.14	1.86	5.8	8.0
2 -Letter Y	74	424	196	62	2.09	2.74	5.7	6.8
3- Letter Z	82	852	161	60	1.96	5.29	10.4	14.2
4- Letter Q	112	1262	146	77	1.30	8.64	11.3	16.4
Hand prep	100	1282	2225	~720	22.25	0.58	12.8	1.5

Table 1: Comparative performance statistics on the use of Ferret to identify the structural components of dictionary records and convert them into a XML marked-up knowledgebase.

restrictions on whether tag sets may occur more than once at a level of nesting are also stored. Provided to the user is the list of tags that can wrap text. These are the names of the semantic fields available to the user for priming. They are taken from the example file.

After the FSA has been constructed, the XML output generator receives a contiguous stream of data/label (tag) pairs. The structure of underlying tags makes it necessary to write the tags in the order given in the example file and is determined by the shortest legal path that nests the current tag in the previous opened tag set. If there is no legal intermediate set of tags to nest the current inside the previous, the previous tag set is closed and the process repeated. This will occur iteratively until a path is found, at which point the open tag and data string are written. An exception to this are <sense> tags sets whose nesting may be determined by an

attribute value corresponding to sense numbers and sub-senses.

Any header and footer information/tags are copied directly from the example file.

10 System Evaluation

The software has been put through some systematic experimentation. It is clear that it functions extremely effectively in terms of its broad goals. It enables appropriate segmentation of entries with semantic tags which are preserved in the PFSA. The software design has proven extremely effective with large number of records being processed with only very short processing delays. All screen functions are completed instantly with no identifiable delays. Table 1 shows performance experiments on 4 files from the same dictionary.

We developed a procedure of starting with the

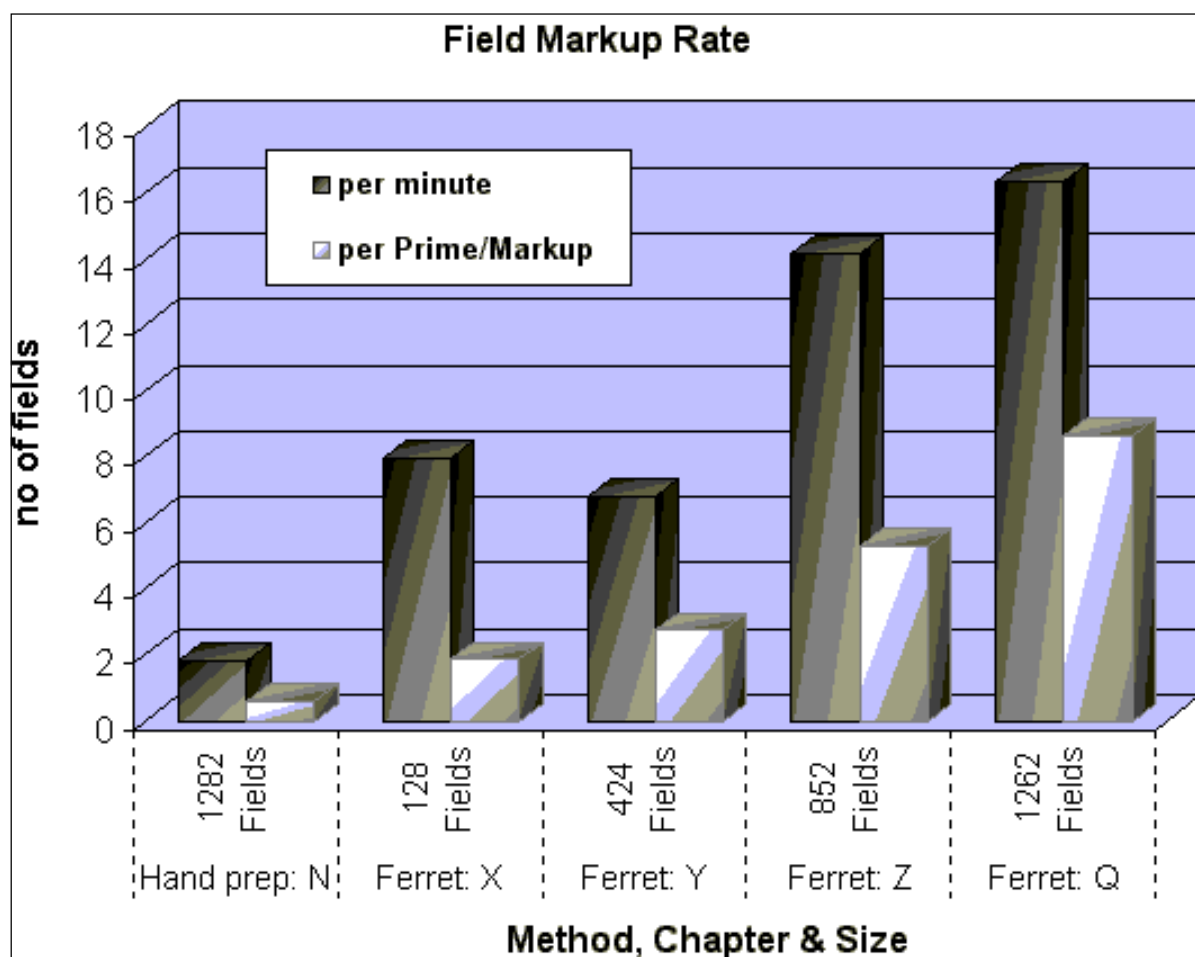


Figure 2. An illustration of the rate at which markup of fields increases in time with progressive training in Ferret by a user on a multi-lingual English-Basque dictionary.

smallest file and working up through the dictionary in ascending order of file size. This enables us to learn about the structure of the files and get acquainted with the mark up complexity of the text in a gradual way. The results from Table 1 indicate significant value in the learning that Ferret III is able to achieve. In the first file 8.0 fields/ min was the priming rate whereas by the time we had completed the fourth file we were working at the rate of 16.4 fields per min, twice the rate. It is noticeable that between the first and second file our priming efficiency dropped from 8.0 to 6.8 fields/min. This was caused principally by a number of records that were very large in length taking a longer than usual amount of time to prime correctly. However the benefit of this work is probably recouped in processing the later files as the third and fourth files have nearly double the fields per record of the first and second files in the ratios of 11.3:10.4:5.7:5.8 (Figure 2). The efficiency of our computational process is demonstrated to be significantly better by an order of magnitude over hand mark-up. Our research assistant who was familiar with the DTD and the tool XMLSpy took 12 hours to prepare the example file of 100 records. For the hand markup, the statistics in Table 1 include structural tags such as <entry>, <form> and <gramGrp>, but excludes header and footer tags. Tag open/close pairs are counted as a single, not two, markups. The number of fields/prime was 0.58 for the hand markup of which 42% of the necessary XML tags for this set were structural, not label tags. The fact that Ferret's output mechanism automatically infers these tags emphasises the further economies it brings to this task.

11 Summary

The system performs to its basic specifications. It is elegantly designed with separation of the data from the inferred automata that represents the breakdown of records into semantic fields. This allows for reuse of an automaton with other data records. Further separation of the interface from the two processing modules improves the maintainability of the system and the quality of the design. The User Interface functions are limited by theoretical considerations of the underlying PFSA model and will require further development. The

tool is usable for the conversion of documents into XML formats according to user-defined semantic fields. Whilst the system may not capture all segmentation desired by a user due to some segmentation not identifiable as RTF tags, users can achieve their semantic segmentation by prior annotation of their text with suitable RTF formatting. Nevertheless Ferret III represents a significant labour saving device compared to total manual mark-up. The system is written in Java to operate under Microsoft Windows.

References

- Ide, N & Veronis, J. (1995). Encoding Dictionaries. *Computers in the Humanities*, **29**: 167-179.
- Turcato, D. Popowich, F., McFetridge, P. & Toole, J. (2000). Creating high-quality, large-scale bilingual knowledge bases using minimal resources. Proc. Workshop on Developing Language Resources for Minority Languages: Reusability and Strategic Priorities. LREC200, Second International Conference on Language Resources and Evaluation. European Language Resources Association. Athens, Greece
- Solomonoff, R.J. (1964). A formal theory of inductive inference. *Information and Control*, **7**, 1-22, 224-258.
- Sperberg-McQueen, C.M., Burnard, L. (1994). Guidelines for the Electronic Text encoding and Interchange, Text Encoding Initiative, Chicago & Oxford.
- Tutin, A. & Veronis, J. (1999). Electronic Dictionary Encoding: Customizing the TEI Guidelines.