FAST-Towards a Semi-automatic Annotation of Corpora

Cătălina Barbu

School of Humanities, Languages and Social Sciences University of Wolverhampton Stafford Road, Wolverhampton, UK in6465@wlv.ac.uk

Abstract

We present in this paper a user-friendly annotation tool that allows a user to perform any kind of annotation on a corpus, either in a manual, semi-automatic or automatic way. We also show how different processing tools can be integrated in the system in order to speed up the human annotation and we describe two such tools that we have integrated in FAST.

1. Introduction

As the use of annotated corpora in natural language processing applications increases, we are aware of the necessity of having flexible annotation tools that would not only support the manual annotation, but also enable us to perform post-editing on a text which has already been automatically annotated using a separate processing tool and even to interact with the tool during the annotation process.

In practice, we have been confronted with the problem of converting the output of different tools to SGML format, while preserving the previous annotation, as well as with the difficulty of post-editing manually an annotated text. It has occurred to us that designing an interface between an annotation tool and any automatic tool would not only provide an easy way of taking advantage of the automatic annotation but it would also allow an easier interactive manual editing of the results. FAST was designed as a manual tagger that can also be used in conjunction with automatic tools for speeding up the human annotation.

2. Corpus annotation

In its modern acceptance, a corpus represents a representative collection of texts in a machine-readable format.

Corpora may take two forms: plain text (unannotated), or annotated (enriched with different kinds of information).

Though unannotated corpora are important for language studies, usability increases by the existence of annotation.

This not only provides additional text information, but it also allows an easier automatic manipulation. An annotated corpus becomes an important repository of linguistic information and so it can be used in many more circumstances then raw text corpora.

2.1. Annotation standards

Though we cannot speak about a generally agreed standard for corpus annotation, there are recent initiatives that aim to provide a standard for the information encoding in electronic texts. The most important trend of this kind is the TEI (Text Encoding Initiative) project, which came with a set of guidelines of how a large number of annotation types can be encoded in an electronic format (Sperberg, 1994). The document markup that TEI uses is SGML. Compared with other annotation methods, encoding the annotation in SGML format presents some advantages. First, it is a widely used marking standard, strongly formalised and easily readable; the information is easy to process automatically and it is also easy to extract separate annotated features or the whole annotation apart from the text for storing in an other format (for example, in a database).

As a result of the need of providing a set of standards for encoding corpora, the European projects MULTEXT and EAGLES, have developed a Corpus Encoding Standard (CES), conformant with the TEI guidelines, for use in language engineering applications, which can serve as a widely accepted set of encoding standards for corpusbased work. CES identifies a minimal encoding level that corpora must achieve to be considered standardized in terms of descriptive representation (marking of structural and linguistic information) as well as general architecture (so as to be maximally suited for use in a text database). It also provides encoding conventions for more extensive encoding and for linguistic annotation. CES favors the storage of the annotation in SGML format (in the new XCES version, in XML format) apart from the main document, to which it is related by hyper-links.

2.2. Types of annotation

Typically, corpora have to contain a basic annotation about the type of the document. Other basic marks, which are considered minimal, are the paragraph and sentence boundaries. Further annotation depends on the purpose for which the annotation is performed.

In order to be used in a natural language processing task, a corpus needs to contain more detailed information about the language structure.

The most usual annotation of this kind is the marking of linguistic information, i.e. assign to words special codes that describe different features. The fundamental linguistic annotation is the part-of-speech tagging, which mainly consist in assigning to each word the morphological category to whom it belongs and specific morphological features like number, gender, case, tense, and so on. This type of annotation is extremely useful in any kind of information extraction system and it also constitutes the basis for most of the natural language processing systems, like syntactic parsing, anaphora resolution. automatic translation, and semantic disambiguation. A higher level of annotation can consist of the marking of stylistic, pragmatic and semantic features.

2.3. Manual annotation vs. automatic annotation

The annotation can be performed manually, by human annotators (one or more persons) or automatically (with or without human post-editing).

The method chosen for annotating a corpus has to establish a balance between the cost (both human and material) involved for building the corpus and the accuracy required.

If only limited funding is available, an error margin of up to 6-7% might be considered acceptable, especially if the tagged corpus is seen as an end product. However, it has to be kept in mind that an important feature of a corpus is its reusability, so existing tags can be used for performing further annotation. In this case, errors in one kind of annotation could carry errors in further types of annotation, and the overall accuracy of the corpus may decrease.

Types of annotation suitable for automatic annotation are for example the morphological annotation. First, because manual annotation of a corpus containing several millions of words is a very expensive and time-consuming process and second, because impressive accuracy has been obtained by different part-of-speech taggers. CLAWS system (Garside, 1987), for example, reports a precision of 96-98%, with different tag sets (of 130-160 tags), while Tosca parser reports a precision of 90-94% (with a tagset of 270 tags). Therefore, the human effort needed for manual post-editing of the tagged corpus is considerably lower than for fully manual annotation.

But for the marking of linguistic features that are intrinsically complex, as it is the case with the anaphoric relations, the semantic or the pragmatics of the text, no tool can perform well enough as to rely on a fully automatic annotation.

3. Annotation tools

Since fully automatic corpus tagging is not yet plausible, the need for low-level annotation tool that can help the human expert in different phases of corpus annotation will always exist.

Currently, there are several annotation tools, more or less designed for specialised tasks. The most common tagger in use is Alembic Workbench, developed at Mitre Corporation, with versions for both Windows and Unix platforms. It allows, besides the mark-up operations for supporting a manual annotation of texts, some other useful operations to be performed, as a scorer, a segmentation tagger, an auto and auto-confirm mode. Despite these functionalities, Alembic is not always intuitive, nor easy to use by a beginner, and a big part of the features existent in the Unix version are not currently available in the Windows version.

Gloss (Cristea, 1998) is a multi-document Windows application that besides the usual textual annotation allows in a friendly way the construction of discourse trees; it also produces a database image of the SGML annotated file. Unfortunately, it is fairly unstable, it only runs under Windows and it is dependant of a manually written DTD. Other annotators for more specific tasks are Xanadu (Fligelstone, 1992), an X-Window editor developed for annotating anaphoric relations within the UCREL anaphoric annotation scheme, CLinkA (Orasan, 2000), developed for co-referential tagging using the MUC annotation scheme, RSTTool (O'Donnell, 1997), for marking the rhetorical structure of a text, AnnoTag, developed at DFKI and used within the VERBMOBIL project for the annotation of spoken dialogues.

Within the MATE project, it is under development a workbench for corpus annotation with extended functionality that will support the use of the standard and enable annotation, different presentation formats, information extraction, statistical analysis, and mapping between different formats.

4. An integrated approach to annotation

As previously stated, an important characteristic of a corpus is its reusability and its ability of being enriched with new kinds of annotation.

The problem with using automatic processing tools is that there is no generally agreed annotation standard, which all of them could employ. This could give raise to several problems concerning the compatibility between two different annotations. First, the output of two processing tools might not be compatible; this not only complicates the annotation process and the post-editing, but it also makes the annotation retrieving more difficult. Second, a processing tool may not recognise as input an already annotated file, or, if it accepts it as input, it may not preserve the existing annotation.

The way of constructing a multi-level annotation on a corpus is to decide on an annotation scheme to be employed and to convert the output of the processing tools to a format consistent with the annotation scheme. There still remains the problem of preserving the previous annotation when applying a new tool, which requires additional pre-processing.

5. Description of Fast

FAST (Friendly Annotator for SGML Texts) is an SGML-compliant tool, which reads any correctly constructed SGML-file, by identifying the tags and the attributes associated to each tag. It uses the Hex-0.9 XML parser (Kristensen), which implements both an event-based–SAX, and a tree-based –DOM- API. We have slightly modified the parser to suit our purposes.

The computer support is provided by an option that allows the user to select one of the three possible working modes: manual (non-automatic), semi-automatic or fully automatic.

5.1. Manual annotation

The non-automatic mode enables the user to manually annotate a text by the way of FAST visual interface. The basic facilities provided by FAST are similar with those offered by other annotation tools. The user can retrieve the information associated with a section of text and can modify it. This includes modifying the name of the tags and the attributes and the values associated with a tag. One can also define new tags and attribute-value pairs for the tags.

A special attention is given to the attributes that can have as associated value an other element of text. In this case, the link is established by the way of the identifier associated to every marked entity.

Annotating a span of text can be done either by selecting the appropriate tag in a combo box, by typing the name of the tag or by pressing the button associated to the tag. By default, the tag that is assigned to a portion of text is the last tag that was used.

There can also be introduced empty tags, which are saved at the end of the SGML document.

5.2. Automatic annotation

What it has to be mentioned from the beginning is that FAST is not an integrated automatic tool for corpus annotation, i.e. it does not provide software for automatic annotation, nor is dependant of any processing tool. It simply allows the user to incorporate already existing tools into the annotator without the necessity of recompiling the application's source or of knowing details about the internal structure of FAST.

The way an aiding tool is integrated in the system is by writing an interface between the output of the abovementioned tool and FAST. This interface is compiled separately and the resulting file is copied in the FAST directory structure. FAST becomes automatically aware of the existence of the tool, loads it dynamically and it can therefore use it when requested. If the tool was for some reason wrongly designed or compiled, or if it is no longer compatible with the current version of FAST, a runtime error is thrown and the tool is not integrated in the program.

Being completely independent of the annotator, the supporting modules can be easily improved or modified in any way without affecting the implementation of the whole project. The only way an automatic tool can reduce the functionality of FAST is if the tool is platform-dependant – in which case it not be run directly from FAST, if running under a different environment.

Semi-automatic annotation can be used for two purposes:

• Correcting the annotation errors introduced by the automatic tool

• Eliminating ambiguities, if the automatic tool allows them to appear.

The semi-automatic mode enables the user to interact with the automatic annotator, being prompted before an annotation is performed; the user can then chose to accept the changes, ignore them or edit the annotation. In the fully automatic mode, the annotation is performed without human intervention. If the automatic tool used allows ambiguities, the most highly ranked alternative is kept. Switching to fully automatic or semi-automatic annotation is realized by selecting from the main menu the tool one wants to use and by customizing its behaviour.

The information that the user can customize by the way of the visual interface consists of the tags that the aiding tool introduces and the attributes associated to each tag. The tag set provided by an aiding tool can be restricted, which is important for example if the user only wants to keep certain type of information (for example, only the words mark-up provided by a morphological parser, not the paragraph or the sentence mark-up, if existent). Similarly, one can eliminate some of the attributes contained in a tag definition. The name of the tags and attributes can also be changed. The default value is the one specified in the aiding tool interface.

An other important feature that can be customized is the behaviour of the tool when other tags are encountered, for example when a level of annotation is added upon older levels. If the text is already annotated, and the tag WORD is present, for example, and a further part-ofspeech annotation is performed that also introduces the WORD tag, the default action would be to construct a wrapping WORD tag around the existing one. This is usually not desirable, since it leads to difficulties in readability; therefore, the user can chose between constructing a wrapping tag and adding the attributes added by the currently used tool to the existent tag. A similar type of behaviour can be customized for the attributes associated to a tag; if they already exist in the annotation, the user can chose to either keep the previous value or replace it with the new one.

5.3. Defining a DTD

Though theoretically it is possible to annotate without specifying a DTD, it seems that most of the time it is advisable to define a configuration file for a specific task. There are at least two evident reasons for defining a DTD: first, the risk of human errors is reduced, because the correctness and consistency of the annotation can be insured through the visual interface; second, the annotation process is easier and less time-consuming, since there are only a limited number of tags to select from and a limited number of attributes.

The idea behind FAST is to allow both a DTD-free and a DTD-based annotation. The user can define a DTD in an interactive, easy-to-use mode; the information required consists of the tagset used and the relationships that can appear between tags. This information can be stored either as a default configuration file (that will be loaded every time a document is open in FAST, which is useful when there is one kind of annotation that is predominantly used), or as a simple configuration file. At any moment, the user can load an other DTD stored in a file.

Though the system ensures that the annotation is consistent with the defined DTD, if the SGML file is modified outside FAST it is not possible to determine inconsistencies in annotation at loading time. This happens because an SGML file is not associated with a certain DTD file.

5.4. Visual interface

FAST uses a friendly visual interface, which was designed to help the user to carry out the annotation task in an efficient way and to enable them to visualise the result of their annotation at any moment. The principle behind the interface was to minimise the number of keystrokes necessary for performing an operation. Both the mouse and the keyboard can be used as input.

It is possible to visualise the tags by using only the mouse, and it is also reasonably easy to add new tags.

The main window is split in two resizable parts. The text to be annotated is displayed in the left, while information about the currently visualised/edited tag are displayed in the right part of the window.

The bottom panel contains a list of buttons, each of them associated with a different tag; their default action is to construct a new tag of the type indicated by the button for the segment of text currently selected.

Every time a new part of text is annotated, it is highlighted in the editor window, and if a new tag was introduced, a new button associated to the tag is added in the bottom panel. The user can visualize the tags existent in the system. The tags that are associated with a segment of text are displayed by highlighting the text, while the extra-textual tags are displayed in a separate window.

In the semi-automatic mode, the results are displayed and the user's input is required in a separate window, in order to avoid confusion.

5.5. Implementation

It is worth mentioning the importance of portability for such a tool, featuring platform independence as well as language independence. We have chosen to develop FAST in Java, using JDK 1.2, so it takes advantage of the Java platform independence features, as well as of the Unicode support, which allows for documents using any set of characters to be processed.

The structure of FAST is modular, allowing easy extension and code reusability.

6. Structure of an aiding tool

The functionality of a supporting module is implemented via a Java API (Application Programming Interface), which defines the basic behaviour of any tool.

The main information that an aiding tool has to provide consists in: its name (the name that appears in the FAST menu bar), the tags that it adds or modifies and, for each modified tag, the attributes that it adds or modifies.

For every span of text that is annotated, it has to specify the text, the tag that is added and the possible values for every attribute.

The design of the interface is based on the assumption that a processing tool introduces only one level of annotation, meaning that we cannot have two different tags applied on the very same segment of text.

It is important to mention that the way the interface between FAST and the supporting module is designed can differ very much from case to case.

For example, if the implementation details of the module are not known, the interface can be seen as a pipe process between the output of the tool and FAST. The file currently open in FAST is submitted to the tool for processing and the resulting file is then parsed and the information transmitted back to FAST.

If the implementation of the tool is known and accessible, it should be possible to transmit the result to FAST without the need for an intermediary output file.

Basically, the Java interface that one has to implement in order to integrate an automatic tool in the visual environment provided by FAST consists in functions for retrieving the information mentioned above. In addition, the user has to define the behaviour of the tool when the next marked element is encountered.

7. Two annotation tools

Because it was originally designed as a tool for annotating anaphoric relations in corpora, FAST currently has some features that are more useful for this particular task than for any other kind of annotation. Two tools have been developed for sustaining the anaphoric tagging, and a third one (for providing WordNet access) is under development.

7.1. Part of speech tagging

The first tool that we have linked to FAST is QTag, the probabilistic part-of-speech tagger developed at the University of Birmingham (Mason).

The way we implemented the interface between QTag and FAST was by the mean of parsing the output file of the tagger.

The parser can be used in either an automatic or semiautomatic way. As it is fairly accurate, the post-editing of the results is not very expensive. Since QTAG is a nonrobust tool, it proposes a list of possible values for each attribute, so it was really useful to have the semiautomatic feature of FAST for choosing between alternatives.

7.2. Coreferential annotation

The second tool we have developed is a module that provides the human editor with hints regarding the preferred antecedents for a noun phrase.

The module was definitely not designed for automatic annotation of anaphoric relations, but as a basic aid for human annotators.

Though it still relies mainly on human intervention, the annotation of co-referential links is improved by a very simple, but fast process that analyses the morphological features of the noun phrases and returns for each referential expression a list of possible antecedents based on morphological agreement.

We have used the QTag part-of-speech tagger trained for English, so obviously the process is limited to analysing anaphoric relations in this language. The results of any part-of-speech tagger are not 100% accurate, and the co-referential relations do not always obey the morphological agreement. Therefore it would not be realistic to propose as antecedents only those discourse entities that pass this basic filter; the final decision belongs to the user, who can reject the proposed antecedents and select any other discourse entity.

The module interacts directly with FAST structure, so there is no need for an intermediary output file.

8. Conclusion & Future Work

We have presented in this paper FAST, a visual annotation tool that provides support for human annotation. We have also showed how an interface can be designed between different automatic annotation tools can FAST for the user to be able to perform interactive editing of the results.

As FAST is still under development, a lot of ideas have yet to be implemented. In the future, we would like to be able to automatically deduce the structure of an SGML input file, and to develop more support modules, especially for the annotation of anaphoric relations. An other prospective feature is the conversion between annotation schemes.

We would also like to improve the user interface, by providing an undo feature, and enabling the user to define their own shortcut keys for different actions.

	r		
Dean Foods Will Buy Birds Eye Brand	Tag		
From Philip Norris Unit for #140 Million	Attributes:		
By Richard Gibson	New attribute:	Value:	
Staff Reporter of The Wall Street Journal			Add
Dean Foods Co., already the country's	Save tag	Delete t_	Clear
leading processor of private-label frozen vegetables, agreed to acquire the Birds Eye	- sover targ		Casa
frozen vegetables brand from Philip Morris Cos.'			
Kraft General Foods			
for \$140 million.			
If approved by federal regulators both the			
company and industry analysts say they expect no			

Figure 1: A snapshot of FAST during a manual annotation process

9. References

- Cristea, D., Craciun, O. & Ursu, C, 1998. Gloss-a Visual Annotation Tool for Corpus Annotation, Essili'98, Saarbruecken
- Fligelstone, S., 1992. *Developing a Scheme for Annotating Text to Show Anaphoric Relations*. In: G. Leitner (ed.), *New Directions in Corpus Linguistics*. Berlin: Mouton de Gruyter. pp153-170.
- Garside, R., 1987. The CLAWS Word-tagging System. In: R. Garside, G. Leech and G. Sampson (eds), The Computational Analysis of English: A Corpus-based Approach. London: Longman.
- Kristensen, A. HEX The HTML Enabled XML Parser, http://hplbwww.hpl.hp.com/people/ak/java/hex.html

Leech, G., 1993. *Corpus annotation schemes*, Literary and Linguistic Computing 8(4): 275-81.

- Mason, O. *QTag, A Portable Part of Speech Tagger.* "http://www-clg.bham.ac.uk/QTAG/"
- O'Donnell, M., 1997. *RSTTool- an RST Analysis Tool.* Proceedings of the 6th European Workshop on Natural Language Generation March 24 - 26, 1997 Gerhard-Mercator University, Duisburg, Germany
- Orasan, C., 2000. ClinkA a Coreferential Links Annotator. forthcoming
- Sperberg-McQueen, C.M. & Burnard, L, 1994. Guidelines for Text Encoding and Interchange (TEI-P3). ACH-ACL-ALLC Text Encoding Initiative, Chicago and Oxford