Automatic Annotation Suggestions and Custom Annotation Layers in WebAnno

Seid Muhie Yimam[1] Richard Eckart de Castilho[2] Iryna Gurevych[2,3] Chris Biemann[1]
(1) FG Language Technology, Dept. of Computer Science, Technische Universität Darmstadt
(2) Ubiquitous Knowledge Processing Lab (UKP-TUDA)
Dept. of Computer Science, Technische Universität Darmstadt
(3) Ubiquitous Knowledge Processing Lab (UKP-DIPF)
German Institute for Educational Research and Educational Information
http://www.{lt,ukp}.tu-darmstadt.de

Abstract

In this paper, we present a flexible approach to the efficient and exhaustive manual annotation of text documents. For this purpose, we extend WebAnno (Yimam et al., 2013) an open-source web-based annotation tool.[1] While it was previously limited to specific annotation layers, our extension allows adding and configuring an arbitrary number of layers through a web-based UI. These layers can be annotated separately or simultaneously, and support most types of linguistic annotations such as spans, semantic classes, dependency relations, lexical chains, and morphology. Further, we tightly integrate a generic machine learning component for automatic annotation suggestions of span annotations. In two case studies, we show that automatic annotation suggestions, combined with our split-pane UI concept, significantly reduces annotation time.

1 Introduction

The annotation of full text documents is a costly and time-consuming task. Thus, it is important to design annotation tools in such a way that the annotation process can happen as swiftly as possible. To this end, we extend WebAnno with the capability of suggesting annotations to the annotator.

A general-purpose web-based annotation tool can greatly lower the entrance barrier for linguistic annotation projects, as tool development costs and preparatory work are greatly reduced. WebAnno 1.0 only partially fulfilled desires regarding generality: Although it covered already more kinds of annotations than most other tools, it supported only a fixed set of customizable annotation layers (named entities, part-of-speech, lemmata, coreference, dependencies). Thus, we also remove a limitation of the tool, which was previously bound to specific, hardcoded annotation layers.

We have generalized the architecture to support three configurable generic structures: *spans*, *relations*, and *chains*. These support all of the original layers and allow the user to define arbitrary custom annotation layers based on either of these structures. Additionally, our approach allows maintaining multiple properties on annotations, e.g. to support morphological annotations, while previously only one property per annotation was supported.

Automatic suggestion of annotations is based on machine learning, which is common practice in annotation tools. However, most of existing web-based annotation tools, such as GATE (Cunningham et al., 2011) or brat (Stenetorp et al., 2012), depend on external preprocessing and post-processing plugins or on web services. These tools have limitations regarding adaptability (difficulty to adapt to other annotation tasks), reconfigurability (generating a classifier when new features and training documents are available is complicated), and reusability (requires manual intervention to add newly annotated documents into the iteration).

For our approach, we assume that an annotator actually does manually verify all annotations to produce a completely labeled dataset. This task can be sped up by automatically suggesting annotations that the annotator may then either accept or correct. Note that this setup and its goal differs from an active learning scenario, where a system actively determines the most informative yet unannotated example to be labeled, in order to quickly arrive at a high-quality classifier that is then to be applied to large amounts of unseen data.

Our contribution is the integration of machine learning into the tool to support exhaustive an-

[1] WebAnno is open-source software under the terms of the Apache Software License 2.0. This paper describes v1.2: http://webanno.googlecode.com

notation of documents providing a shorter loop than comparable tools (Cunningham et al., 2011; Stenetorp et al., 2012), because new documents are added to the training set as soon as they are completed by the annotators. The machine learning support currently applies to sequence classification tasks only. It is complemented by our extension allowing to define custom annotation layers, making it applicable to a wide range of annotation tasks with only little configuration effort.

Section 2 reviews related work about the utilization of automatic supports and customization of annotation schemes in existing annotation tools. The integration of automatic suggestions into WebAnno, the design principles followed, and two case studies are explained in Section 3. Section 4 presents the implementation of customizable annotation layers into the tool. Finally, Section 5 summarizes the main contributions and future directions of our work.

2 Related Work

Automatic annotation support The impact of using lexical and statistical resources to produce pre-annotation automatically to increase the annotation speed has been studied widely for various annotation tasks. For the task of medical named entity labeling, Lingren et al. (2013) investigate the impact of automatic suggestions on annotation speed and potential biases using dictionary-based annotations. This technique results in 13.83% to 21.5% time saving and in an inter-annotator agreement (IAA) increase by several percentage points.

WordFreak (Morton and LaCivita, 2003) includes an automation component, where instances with a low machine learning confidence are presented for annotation in an active learning setup. Beck et al. (2013) demonstrate that the use of active learning for machine translation reduces the annotation effort and show a reduced annotation load on three out of four datasets.

The GoldenGATE editor (Sautter et al., 2007) integrates NLP tools and assistance features for manual XML editing. The tool is used in correcting/editing an automatically annotated document with an editor where both text and XML markups are modified. GoldenGATE is merely used to facilitate the correction of an annotation while pre-annotation is conducted outside of the tool.

Automatic annotation support in brat (Stenetorp et al., 2012) was carried out for a semantic class disambiguation task to investigate how such automation facilitates the annotators' progress. They report a 15.4% reduction in total annotation time. However, the automation process in brat 1) depends on bulk annotation imports and web service configurations, which is labor intensive, 2) is task specific so that it requires a lot of effort to adapt it to different annotation tasks, 3) there is no way of using the corrected result for the next iteration of training the automatic tool.

The GATE Teamware (Bontcheva et al., 2013) automation component is most similar to our work. It is based either on plugins and externally trained classification models, or uses web services. Thus, it is highly task specific and requires extensive configuration. The automatic annotation suggestion component in our tool, in contrast, is easily configurable and adaptable to different annotation tasks and allows the use of annotations from the current annotation project.

Custom annotation layers Generic annotation data models are typically directed graph models (e.g. GATE, UIMA CAS (Götz and Suhre, 2004), GrAF (Ide and Suderman, 2007)). In addition, an annotation schema defines possible kinds of annotations, their properties and relations. While these models offer great expressiveness and flexibility, it is difficult to adequately transfer their power into a convenient annotation editor. For example, one schema may prescribe that the part-of-speech tag is a property on a *Token* annotation, another one may prescribe that the tag is a separate annotation, which is linked to the token. An annotator should not be exposed to these details in the UI and should be able to just edit a part-of-speech tag, ignorant of the internal representation.

This problem is typically addressed in two ways. Either, the full complexity of the annotation model is exposed to the annotator, or the annotation editor uses a simplified model. The first approach can easily lead to an unintuitive UI and make the annotation an inconvenient task. The second approach (e.g. as advocated by brat) requires the implementation of specific import and export filters to transform between the editor data model and the generic annotation data models.

We propose a third approach integrating a configurable mapping between a generic annotation model (UIMA CAS) and a simplified editing model (brat) directly into the annotation tool. Thus, we avoid exposing the full complexity of

the generic model to the user and also avoid the necessity for implementing import/export filters. Similar approaches have already been used to map annotation models to visualization modules (cf. (Zeldes et al., 2009)), but have, to our knowledge, not been used in an annotation editor. Our approach is different from schema-based annotation editors (e.g. GATE), which employ a schema as a template of properties and controlled vocabularies that can be used to annotate documents, but which do not allow to map structures inherent in annotations, like relations or chains, to respective concepts in the UI.

3 Automatic Annotation Suggestions

It is the purpose of the automatic annotation suggestion component to increase the annotation efficiency, while maintaining the quality of annotations. The key design principle of our approach is a split-pane (Figure 1) that displays automatic annotation suggestions in the *suggestion pane* (lower part) and only verified or manual ones in the *annotation pane* (upper part). In this way, we force the annotators to review each automatic suggestion as to avoid overlooking wrong suggestions.

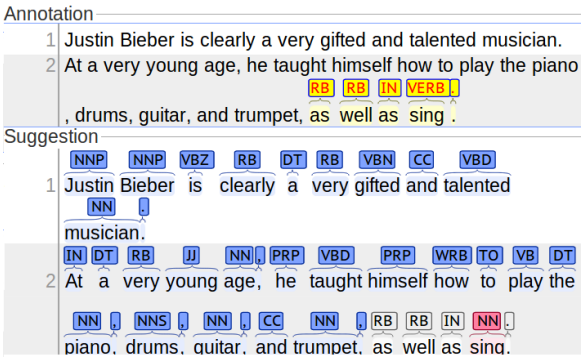

Figure 1: Split-pane UI. Upper: the *annotation pane*, which should be completed by the annotator. Lower: the *suggestion pane*, displaying predictions or automatic suggestions, and coding their status in color. This examples shows automatic suggestions for parts-of-speech. Unattended annotations are rendered in blue, accepted annotations in grey and rejected annotations in red. Here, the last five POS annotations have been attended, four have been accepted by clicking on the suggestion, and one was rejected by annotating it in the annotation pane.

3.1 Suggestion modes

We distinguish three modes of automatic annotation suggestion:

Correction mode In this mode, we import documents annotated by arbitrary external tools and present them to the user in the suggestion pane of the annotation page. This mode is specifically appropriate for annotation tasks where a pre-annotated document contains several possibilities for annotations in parallel, and the user's task is to select the correct annotation. This allows to leverage specialized external automatic annotation components, thus the tool is not limited to the integrated automation mechanism.

Repetition mode In this mode, further occurrences of a word annotated by the user are highlighted in the suggestion pane. To accept suggestions, the user can simply click on them in the suggestion pane. This basic – yet effective – suggestion is realized using simple string matching.

Learning mode For this mode, we have integrated MIRA (Crammer and Singer, 2003), an extension of the perceptron algorithm for online machine learning which allows for the automatic suggestions of span annotations. MIRA was selected because of its relatively lenient licensing, its good performance even on small amounts of data, and its capability of allowing incremental classifier updates. Results of automatic tagging are displayed in the suggestion pane. Our architecture is flexible to integrate further machine learning tools.

3.2 Suggestion Process

The workflow to set up an automatically supported annotation project consists of the following steps.

Importing annotation documents We can import documents with existing annotations (manual or automatic). The *annotation pane* of the automation page allows users to annotate documents and the *suggestion pane* is used for the automatic suggestion as shown in Figure 1. The suggestion pane facilitates accepting correct pre-annotations with minimal effort.

Configuring features For the machine learning tool, it is required to define classification features to train a classifier. We have designed a UI where a range of standard classification features for sequence tagging can be configured. The features include morphological features (prefixes, suffixes, and capitalization), n-grams, and other layers as a feature (for example POS annotation as a feature

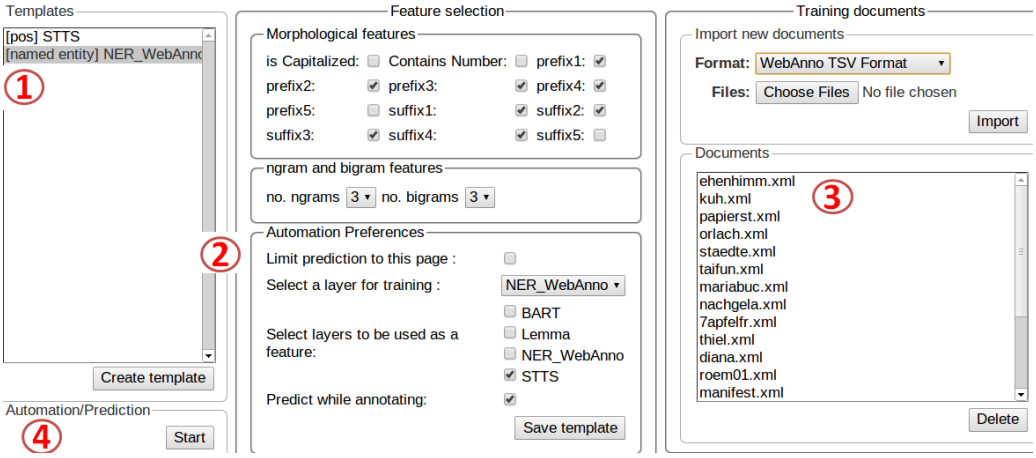

Figure 2: Configuring an annotation suggestion: 1) layers for automation, 2) different features, 3) training documents, 4) start training classifier.

for named entity recognition). While these standard features do not lead to state-of-the-art performance on arbitrary tasks, we have found them to perform very well for POS tagging, named entity recognition, and chunking. Figure 2 shows the feature configuration in the project settings.

Importing training documents We offer two ways of providing training documents: importing an annotated document in one of the supported file formats, such as CoNLL, TCF, or UIMA XMI; or using existing annotation documents in the same project that already have been annotated.

Starting the annotation suggestion Once features for a training layer are configured and training documents are available, automatic annotation is possible. The process can be started manually by the administrator from the automation settings page, and it will be automatically re-initiated when additional documents for training become available in the project. While the automatic annotation is running in the background, users still can work on the annotation front end without being affected. Training and creating a classifier will be repeated only when the feature configuration is changed or when a new training document is available.

Display results on the monitoring page After the training and automatic annotation are completed, detailed information about the training data such as the number of documents (sentence, tokens), features used for each layer, F-score on held-out data, and classification errors are displayed on the monitoring page, allowing an estimation whether the automatic suggestion is useful. The UI also shows the status of the training process (*not started, running, or finished*).

3.3 Case Studies

We describe two case studies that demonstrate language independence and flexibility with respect to sequence label types of our automatic annotation suggestions. In the first case study, we address the task of POS tagging for Amharic as an example of an under-resourced language. Second, we explore German named entity recognition.

3.3.1 Amharic POS tagging

Amharic is an under-resourced language in the Semitic family, mainly spoken in Ethiopia. POS tagging research for Amharic is mostly conducted as an academic exercise. The latest result reported by Gebre (2009) was about 90% accuracy using the Walta Information Center (WIC) corpus of about 210,000 tokens (1065 news documents). We intentionally do not use the corpus as training data because of the reported inconsistencies in the tagging process (Gebre, 2009). Instead, we manually annotate Amharic documents for POS tagging both to test the performance of the automation module and to produce POS-tagged corpora for Amharic. Based upon the work by Petrov et al. (2012) and Ethiopian Languages Research Center (ELRC) tagset, we have designed 11 POS tags equivalent to the Universal POS tags. The tag *DET* is not included as Amharic denotes definiteness as noun suffixes.

We collected some Amharic documents from an online news portal.[2] Preprocessing of Amharic documents includes the normalization of characters and tokenization (sentence and word bound-

[2] http://www.ethiopianreporter.com/

Figure 3: Example Amharic document. The red tags in the suggestion pane have not been confirmed by the annotator.

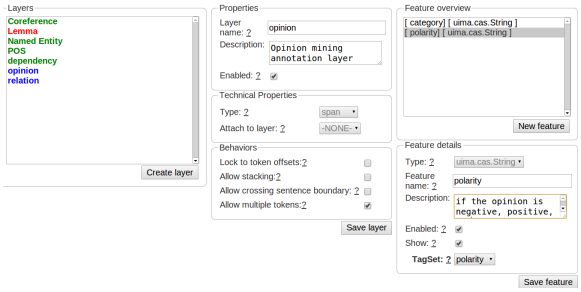

Figure 4: UI for custom annotation layers.

ary detection). Initially, we manually annotated 21 sentences. Using these, an iterative automatic annotation suggestion process was started until 300 sentences were fully annotated. We obtained an F-score of 0.89 with the final model. Hence the automatic annotation suggestion helps in decreasing the total annotation time, since the user has to manually annotate only one out of ten words, while being able to accept most automatic suggestions. Figure 3 shows such an Amharic document in WebAnno.

3.3.2 German Named Entity Recognition

A pilot Named Entity Recognition (NER) project for German was conducted by Benikova et al. (2014). We have used the dataset – about 31,000 sentences, over 41,000 NE annotations – for training NER. Using this dataset, an F-score of about 0.8 by means of automatic suggestions was obtained, which leads to an increase in annotation speed of about 21% with automatic suggestion.

4 Customs Annotation Layers

The tasks in which an annotation editor can be employed depends on the expressiveness of the underlying annotation model. However, fully exposing the expressive power in the UI can make the editor inconvenient to use.

We propose an approach that allows the user to configure a mapping of an annotation model to concepts well-supported in a web-based UI. In this way, we can avoid to expose all details of the annotation model in the UI, and remove the need to implement custom import/export filters.

WebAnno 1.0 employs a variant of the annotation UI provided by brat, which offers the concepts of *spans* and *arcs*. Based on these, WebAnno 1.2 implements five annotation layers: *named entity*, *part-of-speech*, *lemmata*, *co-reference*, and *dependencies*. In the new WebAnno version, we generalized the support for these five layers into three structural categories: *span*, *relation* (arc), and *chain*. Each of these categories is handled by a generic *adapter* which can be configured to simulate any of the original five layers. Based on this generalization, the user can now define custom layers (Figure 4).

Additionally, we introduced a new concept of constraints. For example, NER spans should not cross sentence boundaries and attach to whole tokens (not substrings of tokens). Such constraints not only help preventing the user from making invalid annotations, but can also offer extra convenience. We currently support four hard-coded constraints:

Lock to token offsets Defines if annotation boundaries must coincide with token boundaries, e.g. named entities, lemmata, part-of-speech, etc. For the user's convenience, the annotation is automatically expanded to include the full token, even if only a part of a token is selected during annotation (span/chain layers only).

Allow multiple tokens Some kinds of annotations may only cover a single token, e.g. part-of-speech, while others may cover multiple tokens, e.g. named entities (span/chain layers only).

Allow stacking Controls if multiple annotations of the same kind can be at the same location, e.g. if multiple lemma annotations are allowed per token. For the user's convenience, an existing annotation is replaced if a new annotation is created when stacking is not allowed.

Allow crossing sentence boundaries Certain annotations, e.g. named entities or dependency relations, may not cross sentence boundaries, while others need to, e.g. coreference chains.

Finally, we added the ability to define multiple properties for annotations to WebAnno. For example, this can be use to define a custom span-based *morphology* layer with multiple annotation properties such as *gender*, *number*, *case*, etc.

5 Conclusion and Outlook

We discussed two extensions of WebAnno: the tight and generic integration of automatic annotation suggestions for reducing the annotation time, and the web-based addition and configuration of custom annotation layers.

While we also support the common practice of using of external tools to automatically pre-annotate documents, we go one step further by tightly integrating a generic sequence classifier into the tool that can make use of completed annotation documents from the same project. In two case studies, we have shown quick convergence for Amharic POS tagging and a substantial reduction in annotation time for German NER. The key concept here is the split-pane UI that allows to display automatic suggestions, while forcing the annotator to review all of them.

Allowing the definition of custom annotation layers in a web-based UI is greatly increasing the number of annotation projects that potentially could use our tool. While it is mainly an engineering challenge to allow this amount of flexibility and to hide its complexity from the user, it is a major contribution in the transition from specialized tools towards general-purpose tools.

The combination of both – custom layers and automatic suggestions – gives rise to the rapid setup of efficient annotation projects. Adding to existing capabilities in WebAnno, such as curation, agreement computation, monitoring and fine-grained annotation project definition, our contributions significantly extend the scope of annotation tasks in which the tool can be employed.

In future work, we plan to support annotation suggestions for non-span structures (arcs and chains), and to include further machine learning algorithms.

Acknowledgments

The work presented in this paper was funded by a German BMBF grant to the CLARIN-D project, the Hessian LOEWE research excellence program as part of the research center "Digital Humanities" and by the Volkswagen Foundation as part of the Lichtenberg-Professorship Program under grant No. I/82806.

References

Daniel Beck, Lucia Specia, and Trevor Cohn. 2013. Reducing annotation effort for quality estimation via active learning. In *Proc. ACL 2013 System Demonstrations*, Sofia, Bulgaria.

Darina Benikova, Chris Biemann, and Marc Reznicek. 2014. NoSta-D Named Entity Annotation for German: Guidelines and Dataset. In *Proc. LREC 2014*, Reykjavik, Iceland.

Kalina Bontcheva, H. Cunningham, I. Roberts, A. Roberts, V. Tablan, N. Aswani, and G. Gorrell. 2013. GATE Teamware: a web-based, collaborative text annotation framework. *Language Resources and Evaluation*, 47(4):1007–1029.

Koby Crammer and Yoram Singer. 2003. Ultraconservative online algorithms for multiclass problems. In *Journal of Machine Learning Research 3*, pages 951 – 991.

Hamish Cunningham, D. Maynard, K. Bontcheva, V. Tablan, N. Aswani, I. Roberts, G. Gorrell, A. Funk, A. Roberts, D. Damljanovic, T. Heitz, M. A. Greenwood, H. Saggion, J. Petrak, Y. Li, and W. Peters. 2011. *Text Processing with GATE (Version 6)*. University of Sheffield Department of Computer Science, ISBN 978-0956599315.

Binyam Gebrekidan Gebre. 2009. Part-of-speech tagging for Amharic. In *ISMTCL Proceedings, International Review Bulag, PUFC*.

T. Götz and O. Suhre. 2004. Design and implementation of the UIMA Common Analysis System. *IBM Systems Journal*, 43(3):476 –489.

Nancy Ide and Keith Suderman. 2007. GrAF: A graph-based format for linguistic annotations. In *Proc. Linguistic Annotation Workshop*, pages 1–8, Prague, Czech Republic.

Todd Lingren, L. Deleger, K. Molnar, H. Zhai, J. Meinzen-Derr, M. Kaiser, L. Stoutenborough, Q. Li, and I. Solti. 2013. Evaluating the impact of pre-annotation on annotation speed and potential bias: natural language processing gold standard development for clinical named entity recognition in clinical trial announcements. In *Journal of the American Medical Informatics Association*, pages 951 – 991.

Thomas Morton and Jeremy LaCivita. 2003. WordFreak: an open tool for linguistic annotation. In *Proc. NAACL 2003*, demonstrations, pages 17–18, Edmonton, Canada.

Slav Petrov, Dipanjan Das, and Ryan McDonald. 2012. A universal part-of-speech tagset. In *Proc LREC 2012*, Istanbul, Turkey.

Guido Sautter, Klemens Böhm, Frank Padberg, and Walter Tichy. 2007. *Empirical Evaluation of Semi-automated XML Annotation of Text Documents with the GoldenGATE Editor*. Budapest, Hungary.

Pontus Stenetorp, Sampo Pyysalo, Goran Topić, Tomoko Ohta, Sophia Ananiadou, and Jun'ichi Tsujii. 2012. brat: a Web-based Tool for NLP-Assisted Text Annotation. In *Proc. EACL 2012 Demo Session*, Avignon, France.

Seid Muhie Yimam, Iryna Gurevych, Richard Eckart de Castilho, and Chris Biemann. 2013. WebAnno: A flexible,web-based and visually supported system for distributed annotations. In *Proc. ACL 2013 System Demonstrations*, pages 1–6, Sofia, Bulgaria.

Amir Zeldes, Julia Ritz, Anke Lüdeling, and Christian Chiarcos. 2009. ANNIS: A search tool for multi-layer annotated corpora. In *Proc. Corpus Linguistics 2009*, Liverpool, UK.