Analyzing and Visualizing Coreference Resolution Errors

Sebastian Martschat[1], **Thierry Göckel**[2] and **Michael Strube**[1]
[1]Heidelberg Institute for Theoretical Studies gGmbH, Heidelberg, Germany
(sebastian.martschat|michael.strube)@h-its.org
[2]iQser GmbH, Walldorf, Germany
thierry.goeckel@iqser.com

Abstract

We present a toolkit for coreference resolution error analysis. It implements a recently proposed analysis framework and contains rich components for analyzing and visualizing recall and precision errors.

1 Introduction

Coreference resolution is the task of determining which mentions in a text refer to the same entity. Both the natural language processing engineer (who needs a coreference resolution system for the problem at hand) and the coreference resolution researcher need tools to facilitate and support system development, comparison and analysis.

In Martschat and Strube (2014), we propose a framework for error analysis for coreference resolution. In this paper, we present *cort*[1], an implementation of this framework, and show how it can be useful for engineers and researchers. *cort* is released as open source and is available for download[2].

2 Error Analysis Framework

Due to the set-based nature of coreference resolution, it is not clear how to extract errors when an entity is not correctly identified. The idea underlying the analysis framework of Martschat and Strube (2014) is to employ spanning trees in a graph-based entity representation.

Figure 1 summarizes their approach. They represent reference and system entities as complete one-directional graphs (Figures 1a and 1b). To extract recall errors, they compute a spanning tree of the reference entity (Figure 1a). All edges in the spanning tree which do not appear in the system output are extracted as recall errors (Figure 1c). For extracting precision errors, the roles of reference and system entities are switched.

The analysis algorithm is parametrized only by the spanning tree algorithm employed: different algorithms lead to different notions of errors. In Martschat and Strube (2014), we propose an algorithm based on Ariel's accessibility theory (Ariel, 1990) for reference entities. For system entity spanning trees, we take each output pair as an edge.

3 Architecture

Our toolkit is available as a Python library. It consists of three modules: the `core` module provides mention extraction and preprocessing, the `coreference` module implements features for and approaches to coreference resolution, and the `analysis` module implements the error analysis framework described above and ships with other analysis and visualization utilities.

3.1 `core`

All input and output must conform to the format of the CoNLL-2012 shared task on coreference resolution (Pradhan et al., 2012). We employ a rule-based mention extractor, which also computes a rich set of mention attributes, including tokens, head, part-of-speech tags, named entity tags, gender, number, se-

[1]Short for **co**reference **r**esolution **t**oolkit.
[2]`http://smartschat.de/software`

Proceedings of NAACL-HLT 2015, pages 6–10,
Denver, Colorado, May 31 – June 5, 2015.

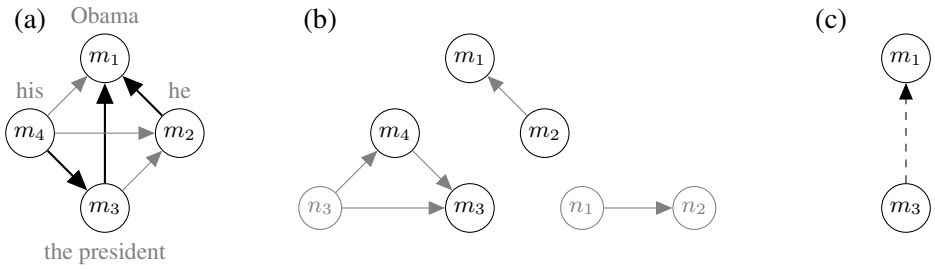

Figure 1: (a) a reference entity r (represented as a complete one-directional graph) and its spanning tree T_r, (b) a set S of three system entities, (c) the errors: all edges in T_r which are not in S.

mantic class, grammatical function, coarse mention type and fine-grained mention type.

3.2 `coreference`

cort ships with two coreference resolution approaches. First, it includes *multigraph*, which is a deterministic approach using a few strong features (Martschat and Strube, 2014). Second, it includes a mention-pair approach (Soon et al., 2001) with a large feature set, trained via a perceptron on the CoNLL'12 English training data.

System	MUC	B^3	$CEAF_e$	Average
StanfordSieve	64.96	54.49	51.24	56.90
BerkeleyCoref	70.27	59.29	56.11	61.89
multigraph	69.13	58.61	56.06	61.28
mention-pair	69.09	57.84	53.56	60.16

Table 1: Comparison of systems on CoNLL'12 English development data.

In Table 1, we compare both approaches with *StanfordSieve* (Lee et al., 2013), the winner of the CoNLL-2011 shared task, and *BerkeleyCoref* (Durrett and Klein, 2013), a state-of-the-art structured machine learning approach. The systems are evaluated via the CoNLL scorer (Pradhan et al., 2014).

Both implemented approaches achieve competitive performance. Due to their modular implementation, both approaches are easily extensible with new features and with training or inference schemes. They therefore can serve as a good starting point for system development and analysis.

3.3 `analysis`

The core of this module is the `ErrorAnalysis` class, which extracts and manages errors extracted from one or more systems. The user can define own spanning tree algorithms to extract errors. We already implemented the algorithms discussed in Martschat and Strube (2014). Furthermore, this module provides functionality to

- categorize and filter sets of errors,
- visualize these sets,
- compare errors of different systems, and
- display errors in document context.

Which of these features is interesting to the user depends on the use case. In the following, we will describe the popular use case of improving a coreference system in detail. Our system also supports other use cases, such as the cross-system analysis described in Martschat and Strube (2014).

4 Use Case: Improving a Coreference Resolution System

A natural language processing engineer might be interested in improving the performance of a coreference resolution system since it is necessary for another task. The needs may differ depending on the task at hand: for some tasks proper name coreference may be of utmost importance, while other tasks need mostly pronoun coreference. Through model and feature redesign, the engineer wants to improve the system with respect to a certain error class.

The user will start with a baseline system, which can be one of the implemented systems in our toolkit or a third-party system. We now describe how *cort* facilitates improving the system.

4.1 Initial Analysis

To get an initial assessment, the user can extract all errors made by the system and then make use of the plotting component to compare these errors with the maximum possible number of errors[3].

For a meaningful analysis, we have to find a suitable error categorization. Suppose the user is interested in improving recall for non-pronominal coreference. Hence, following Martschat and Strube (2014), we categorize all errors by coarse mention type of anaphor and antecedent (proper name, noun, pronoun, demonstrative pronoun or verb)[4].

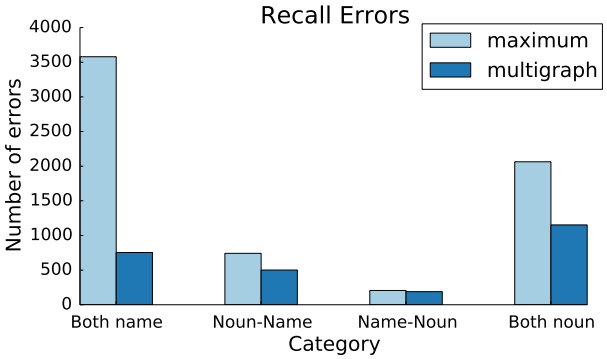

Figure 2: Recall errors of the *multigraph* baseline.

Figure 2 compares the recall error numbers of the *multigraph* system with the maximum possible number of errors for the categories of interest to the engineer. The plot was created by our toolkit via *matplotlib* (Hunter, 2007). We can see that the model performs very well for proper name pairs. Relative to the maximum number of errors, there are much more recall errors in the other categories. A plot for precision errors shows that the system makes only relatively few precision errors, especially for proper name pairs.

After studying these plots the user decides to improve recall for pairs where the anaphor is a noun and the antecedent is a name. This is a frequent category which is handled poorly by the system.

[3]For recall, the maximum number of errors are the errors made by a system which puts each mention in its own cluster. For precision, we take all pairwise decisions of a model.

[4]For a pair of mentions constituting an error, we call the mention appearing later in the text the *anaphor*, the other mention *antecedent*.

4.2 Detailed Analysis

In order to determine how to improve the system, the user needs to perform a detailed analysis of the noun-name errors. Our toolkit provides several methods to do so. First of all, one can browse through the pairwise error representations. This suggests further subcategorization (for example by the presence of token overlap). An iteration of this process leads to a fine-grained categorization of errors.

However, this approach does not provide any document context, which is necessary to understand some errors. Maybe context features can help in resolving the error, or the error results from multiple competing antecedents. We therefore include a visualization component, which also allows to study the interplay between recall and precision.

Figure 3 shows a screenshot of this visualization component, which runs in a web browser using JavaScript. The header displays the identifier of the document in focus. The left bar contains the navigation panel, which includes

- a list of all documents in the corpus,
- a summary of all errors for the document in focus, and
- lists of reference and system entities for the document in focus.

To the right of the navigation panel, the document in focus is shown. When the user picks a reference or system entity from the corresponding list, *cort* displays all recall and precision errors for all mentions which are contained in the entity (as labeled red arrows between mentions). Alternatively, the user can choose an error category from the error summary. In that case, all errors of that category are displayed.

We use color to distinguish between entities: mentions in different entities have different background colors. Additionally mentions in reference entities have a yellow border, while mentions in system entities have a blue border (for example, the mention *the U.S.-backed rebels* is in a reference entity and in a system entity). The user can choose to color the background of mentions either depending on their gold entity or depending on their system entity.

These visualization capabilities allow for a detailed analysis of errors and enable the user to take all document information into account.

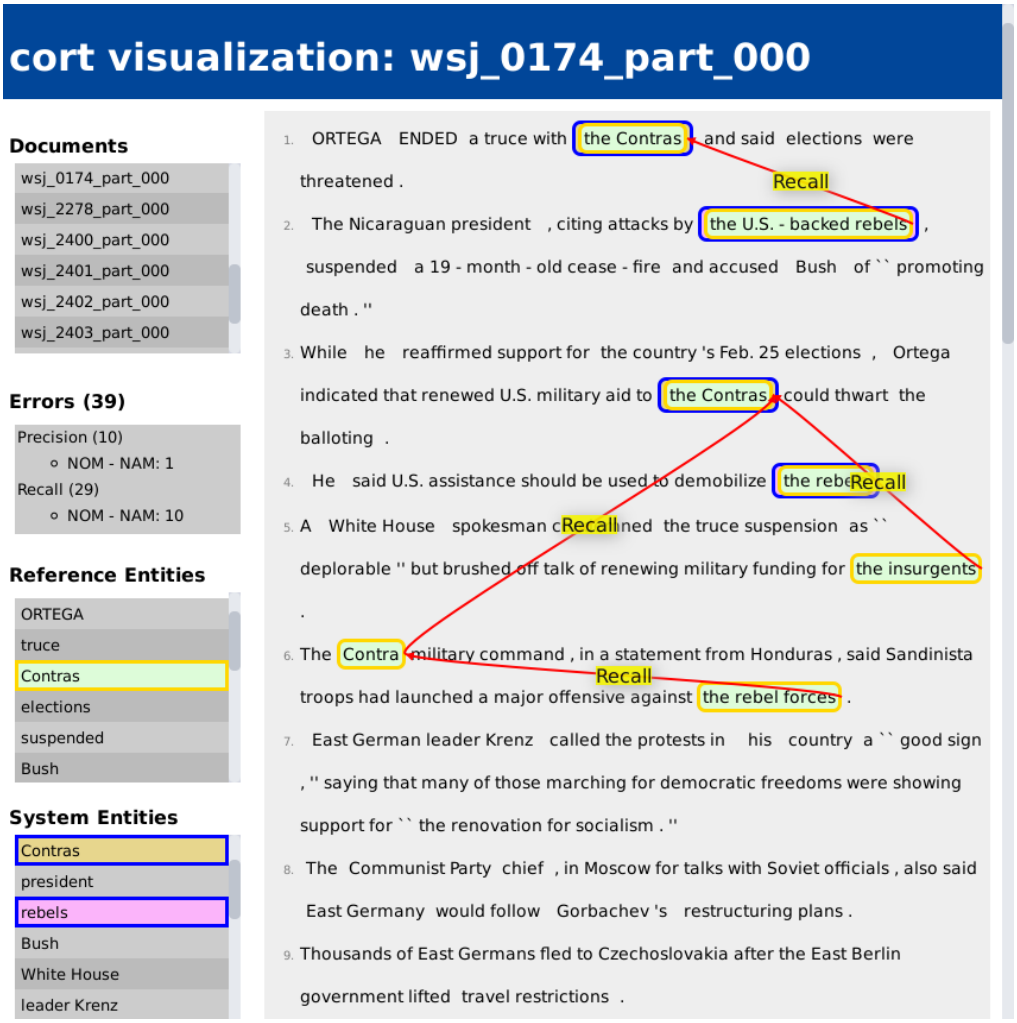

Figure 3: Screenshot of the visualization component.

The result of the analysis is that almost all errors are missed is-a relations, such as in the examples in Figure 3 (*the U.S.-backed rebels* and *the Contras*).

4.3 Error Comparison

Motivated by this, the user can add features to the system, for example incorporating world knowledge from Wikipedia. The output of the changed model can be loaded into the `ErrorAnalysis` object which already manages the errors made by the baseline system.

To compare the errors, *cort* implements various functions. In particular, the user can access common errors and errors which are unique to one or more systems. This allows for an assessment of the qualitative usefulness of the new feature. Depending on the results of the comparison, the user can decide between discarding, retaining and improving the feature.

5 Related Work

Compared to our original implementation of the error analysis framework (Martschat and Strube, 2014), we made the analysis interface more user-friendly and provide more analysis functionality. Furthermore, while our original implementation did not include any visualization capabilities, we now allow for both data visualization and document visualization.

We are aware of two other software packages for coreference resolution error analysis. Our toolkit

complements these. Kummerfeld and Klein (2013) present a toolkit which extracts errors from transformation of reference to system entities. Hence, their definition of what an error is not rooted in a pairwise representation, and is therefore conceptually different from our definition. They do not provide any visualization components.

ICE (Gärtner et al., 2014) is a toolkit for coreference visualization and corpus analysis. In particular, the toolkit visualizes recall and precision errors in a tree-based visualization of coreference clusters. Compared to ICE, we provide more extensive functionality for error analysis and can accommodate for different notions of errors.

6 Conclusions and Future Work

We presented *cort*, a toolkit for coreference resolution error analysis. It implements a graph-based analysis framework, ships with two strong coreference resolution baselines and provides extensive functionality for analysis and visualization.

We are currently investigating whether the analysis framework can also be applied to structurally related tasks, such as cross-document coreference resolution (Singh et al., 2011) or entity linking.

Acknowledgements

This work has been funded by the Klaus Tschira Foundation, Heidelberg, Germany. The first author has been supported by a HITS PhD scholarship.

References

Mira Ariel. 1990. *Accessing Noun Phrase Antecedents*. Routledge, London, U.K.; New York, N.Y.

Greg Durrett and Dan Klein. 2013. Easy victories and uphill battles in coreference resolution. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing,* Seattle, Wash., 18–21 October 2013, pages 1971–1982.

Markus Gärtner, Anders Björkelund, Gregor Thiele, Wolfgang Seeker, and Jonas Kuhn. 2014. Visualization, search, and error analysis for coreference annotations. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations,* Baltimore, Md., 22–27 June 2014, pages 7–12.

John D. Hunter. 2007. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3):90–95.

Jonathan K. Kummerfeld and Dan Klein. 2013. Error-driven analysis of challenges in coreference resolution. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing,* Seattle, Wash., 18–21 October 2013, pages 265–277.

Heeyoung Lee, Angel Chang, Yves Peirsman, Nathanael Chambers, Mihai Surdeanu, and Dan Jurafsky. 2013. Deterministic coreference resolution based on entity-centric, precision-ranked rules. *Computational Linguistics*, 39(4):885–916.

Sebastian Martschat and Michael Strube. 2014. Recall error analysis for coreference resolution. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing,* Doha, Qatar, 25–29 October 2014, pages 2070–2081.

Sameer Pradhan, Alessandro Moschitti, Nianwen Xue, Olga Uryupina, and Yuchen Zhang. 2012. CoNLL-2012 Shared Task: Modeling multilingual unrestricted coreference in OntoNotes. In *Proceedings of the Shared Task of the 16th Conference on Computational Natural Language Learning,* Jeju Island, Korea, 12–14 July 2012, pages 1–40.

Sameer Pradhan, Xiaoqiang Luo, Marta Recasens, Eduard Hovy, Vincent Ng, and Michael Strube. 2014. Scoring coreference partitions of predicted mentions: A reference implementation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers),* Baltimore, Md., 22–27 June 2014, pages 30–35.

Sameer Singh, Amarnag Subramanya, Fernando Pereira, and Andrew McCallum. 2011. Large-scale cross-document coreference using distributed inference and hierarchical models. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers),* Portland, Oreg., 19–24 June 2011, pages 793–803.

Wee Meng Soon, Hwee Tou Ng, and Daniel Chung Yong Lim. 2001. A machine learning approach to coreference resolution of noun phrases. *Computational Linguistics*, 27(4):521–544.