MoReViewer: A Graphical Error Analysis Tool for Morphosyntactic Learning Models

Ioannis Dikeoulias
Saarland University

s9iodike(at)stud.uni-saarland.de

Abstract

Artificial neural networks are one of the most anticipated computing systems of the last decade. Especially natural language processing tasks, such as Machine Translation, Question Answering or Text Simplification benefit from the diverse possibilities which are offered by statistical learning models. But despite the advantages these systems offer, they often rely on huge amount of human-annotated data and need to be validated and evaluated thoughtfully. In this paper we present MoReViewer, a graphical error analysis tool for morphosyntactic learning models that enables both computational linguists as well as non technical scientists to efficiently analyse morphological inflections of any language or grammar.

1 Introduction

Recent challenges in Natural Language Processing (NLP) involve many different methods, such as, speech recognition or language generation which are both from a methodological as well as from a conceptual perspective demanding tasks. Unfortunately, machine learning models often lack of high quality data in order to learn all concepts of a language. As a consequence, they often exploit only specific methodological characteristics of a language and are therefore predestined to be mislead by irregularities and inconsistencies which are not uncommon in linguistics.
In order to recognise potential classification or prediction errors and improve the overall quality of NLP tasks, researchers need to thoroughly investigate the decisions of their machine learning models. Moreover, a sophisticated error analysis requires a variety of statistical tools as well as skilled personnel that is able to correctly interpret the results. In addition, manual error analysis is a time-consuming and expensive task considering the large number of training cycles a model needs to perform in order to be released for evaluation and it requires an adequate amount of motivation and patience. To overcome this problem, many researchers increasingly occupy themselves with the design and development of computer-assisted error analysis (CEA) tools. These tools provide many

functionalities to assist researchers in efficiently validating their statistical models as well as research assistants to contribute effectively to the improvement of these models without the need of any deeper statistical knowledge.
Additionally, error analysis tools help to uncover bugs and reveal common pitfalls within the implementation of statistical models as well as suggest possible opportunities and improvements to increase their overall accuracy and precision of NLP models. To support research in the field of inflectional morphology, this paper introduces a graphical error analysis tool for morphosyntactic learning models. It allows researchers to perform common analysis tasks, such as, searching for specific syntactical patterns, filtering on grammatical features or displaying and comparing useful statistics like the Levenshtein distance.
During the design phase of our tool we followed strict design principles, such as, adding natural feedback mechanisms or integrating control flow restrictions in order to provide an intuitive and satisfiable user experience. In addition, we promote a flexible configuration schema that enables researchers to specify their own grammar within a simple to use JSON format, allowing to analyse model predictions for any language or dialect. In particular, this paper makes the following contributions:

- We review research papers in the domain "Automatic Error Analysis" regarding their design approach and outcomes. (see section 2)
- We present MoReViewer, a graphical error analysis tool for morphosyntactic learning models. (see section 3)
- We critically reflect on the weaknesses and limitations of our proposed design approach in comparison to existing methodologies. (see section 4)
- We identify persistent challenges in the domain of "Computer-aided Error Analysis" for statistical machine learning models and introduce possible topics for future work. (see section 5)

2 Literature Review

NLP tasks have rapidly gained high popularity in many different research areas, such as, Question Answering (QA), Named Entity Recognition (NER) or Optical Character Recognition (OCR), to name but few. As a consequence, graphical error analysis tools and frameworks ([7], [5]) are increasingly promoted and often equipped with graphically-assistive and interactive user interfaces to facilitate researchers everydays life.
Coreference resolution, for example, is one of the most prominent NLP tasks and is used by many researchers within a multitude of application fields, such as, semantic search, machine translation and other information extraction tasks. To facilitate the evaluation of model predictions for coreference resolution tasks, Gaertner et. al. [3] presents the ICARUS Coreference Explorer, an interactive tool to browse and search coreference annotated data. Their tool allows researchers to evaluate errors in system predictions using

a variety of display modes, such as, entity grids or tree views. Further, an integrated search engine for document annotations enables researchers to interactively inspect system errors.
A related and frequently applied task in the field of information retrieval is relation extraction which occupies with the extraction of semantic relationships from text corpora. For this reason, Agarwal et. al. [1] promotes a web-based error analysis tool for both computational linguists as well as non-technical personnel that allows to perform error analysis on different NLP tasks. Within an easy to use framework this tool allows researchers to evaluate their machine learning predictions by providing a XML-formatted version of their model results.
Another work that is contextually related to our work is proposed by the research group of El Kholy et. al. [2]. They present AMEANA, an open source error analysis tool for NLP tasks that concentrates on morphologically rich languages. Using a simple configuration file, it enables researchers to evaluate arbitrary models, independent of the language used. Besides of displaying general statistics, it uses a word alignment algorithm that aligns output words with their matching reference words in order to generate detailed morphological error diagnostics.
Although, error analysis tools are often designed from scratch and provide individual analytical and diagnostic tools to evaluate statistical models, some tools make use of existing libraries that can be visually extended to improve their user experience. Gonzales et. al. [4], for instance, developed a graphically-aided error analysis and diagnosis tool that is able to investigate the strengths and weaknesses of MT systems by automating the detection and classification of errors. Thereby, he introduces an online graphical and interactive interface that visualises information related to evaluation metrics within the ASIYA toolkit. Using different visualisation approaches, such as, constituency and dependency trees, this tool enables to visualise various linguistic measures at the syntactic and semantic level.
Besides of developing standalone helper tools, a frequently used approach to support developers during development is the integration of extensions within integrated development environments. A similar approach is introduced by Mayfield et. al. [6] who presents an interactive error analysis tool for text mining that is built as extension for the integrated development environment SIDE. It is tested within a research environment in conjunction with machine learning projects and allows to identify sentence structures using ML generated filters.

3 Analysis Tool

In this section we present MoReViewer, a computer-aided error analysis tool that offers various simple and easy to use functionalities for the evaluation of machine-generated morphological inflections. To provide a deeper insight into the internal structure and design of this tool, this section is split into two major parts. First, we will focus on the system architecture which includes the technical background of our tool as well as the

system requirements.
In addition, we will introduce our language independent configuration schema and give a short summary of the various functionalities which are provided by our analysis tool. The second part of this section focuses on the interface design which includes the introduction of the different views and display modes as well as the evaluation metrics and statistics that are accessible throughout the MoRe Viewer analysis tool.

3.1 Technical Background

The backend of our graphical error analysis tool MoReViewer is built in Python 3. The user interface is built with PyQt, a Python extension which is based on the popular C++ GUI toolkit Qt. It is freely available for non-commercial use and exists, as of May 2020, in its fifth version. PyQt is cross-platform compatible, supporting the major operating systems, such as Windows, Linux and MacOS and provides over 400 classes and modules which allow developers to quickly build prototypes of their applications. A detailed comparison of the most popular GUI libraries is shown in Table 1.

Table 1: A comparison of Python GUI libraries and frameworks.

	PyQt5	wxPython	Tkinter	Html/JS (Django)
Platform	Windows, Linux, MacOS	Windows, Linux, MacOS	Windows, Linux, MacOS	All (web-based)
Backend	C++	C++ / Python	Python	Python
UI Builder	Yes	Yes	No	No
License	GPL / Commercial	GPL	GPL	BSD

3.2 Problem Statement

The main motivation of this work is to build a tool that allows researchers to quickly spot errors from their machine learning models and reduce the cognitive effort of evaluating model predictions, while speeding up the main development process of statistical learning models. Thus, an important aspect of our tool is a seamless user experience and a user-friendly design in order to facilitate and speed up repetitive tasks during error analysis. Regarding previous approaches, there exist a handful of different error analysis tools (cf. section 2) for NLP tasks that could give us insightful ideas on how to start the development of an error analysis tool. Unfortunately, only one approach (cf. El Kholy et. al.) focused on inflectional morphology that was published nine years ago. Therefore, our group decided to design and develop a graphical error analysis tool from the ground up, focusing on the individual requirements that are demanded by an error analysis tool within the domain of inflectional morphology.

3.3 System Requirements

The requirements of our tool are gathered iteratively over multiple group meetings and reflect the individual requirements of our group members that were revealed during the development of a statistical learning model for morphological reinflections which is based on the model of Wu et. al. [8]. Following, we will list the requirements for our error analysis tool. For a better readability, we divided them into three categories:

Graphical Requirements Graphical requirements cover visual expectations on the analysis tool to guarantee an improved visibility and affordability of all functionalities.

- A list-based visualisation of NLP outputs,
- a comparative error highlighting for prediction/groundtruth pairs,
- a multi-document area to support side-by-side views,
- a full-screen mode with resizable/hideable subwindows and
- a graphical representation of simple NLP statistics.

Analytical Requirements Analytical requirements include functional expectations that are required to provide the necessary tools for the error diagnosis.

- A configuration file to specify format and grammar of NLP outputs,
- a sorting mechanism on all columns,
- a filtering mechanism on grammatical features,
- a visualisation of input/output alignments and their corresponding features,
- a tagging system for each prediction row and
- access to simple evaluation metrics and statistics.

Supportive Requirements Supportive requirements include additional enhancements to various system modules to further enhance the user experience and productivity of our tool.

- An import/export mechanism for NLP outputs,
- an import wizard to guide users through the import process,
- a default grammar to enable a quick visualisation of results,
- a quick access to recent projects,
- a comment section for each prediction row and

- an automatic language detection mechanism for NLP outputs.

3.4 System Architecture

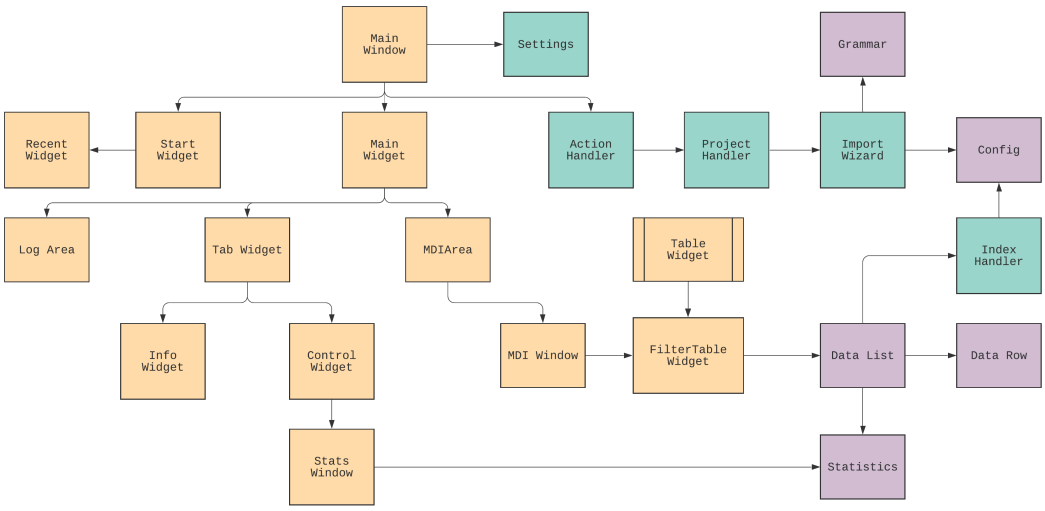


Figure 1: A class diagram of MoReViewer. Classes are coloured according to their MVC-level: model (violet), view (yellow), controller (green).

3.4.1 Schema Configuration

An important requirement of our analysis tool is that it is usable in conjunction with arbitrary language models. For this purpose, we promote two configuration files that are easily adjustable to the users needs. First, during the import of a new project the user needs to configure the result schema of his statistical model within a JSON file named *config.json*.
The result schema provides a simple interface that allows users to freely extend their model outputs by enabling the associated result field under the configuration key *columns*. This way, users can individually adapt their result schema and still benefit from the various functionalities of our analysis tool. Following, we list all supported result fields:

- input: the model input (*required*),
- tags: the grammatical tag set (*required*),
- prediction: the model prediction (*required*),
- groundtruth: the gold standard (*required*),
- alignment: the character alignment string (*optional*) and

- features: a set of alignment features separated by a semicolon (;) (*optional*).

Secondly, the user has the option to specify an individual grammar for the error analysis. Therefore, he needs to configure a *grammar.json* file, that currently supports the following two specifications:

1. grammar: a dictionary of grammatical categories, including a mapping of their features in the form *TAG:FEATURE* and
2. grammar-dependencies: a dictionary of grammatical dependencies of the form *CATEGORY:TAG*, including a set of grammatical categories.

The first specification declares the set of grammatical categories that are supported by specifying their grammatical *TAG* and respective *FEATURE*. The second specification declares which categories should be further editable after the selection of a specific *TAG* within a grammatical *CATEGORY*. Examples for both configuration files can be found in the appendix A.

3.4.2 Character Alignment and Tagging

A common approach in order to gain a detailed understanding on the decisions of a NLP model is the usage of word or character-based alignments, dependent on the underlying model. Word alignment, also used in various related works ([2], [4]) is the task of identifying word relationships among the words of two related texts.
In contrast, character alignment which is further promoted in our analysis tool is the task of identifying character-based relationships between two related words. Our tool implicitly assumes a character alignment between the *input* and *prediction* word. Users can integrate character-alignments by specifying the two fields *aligment* and *features* within the configuration file *config.json*.
For an improved visibility of the character alignment, the *alignment* and *features* are displayed together with the *prediction* within a separate table under the *Info* tab. An example for a character-based alignment is shown below, where "_" denotes the *COPY* process:

Prediction	r	u	n	d	e	n	t	e	n
Alignment	3	4	5	6	7	8	9	9	9
Features	_	_	_	_	_	_	3;PST	IND;PST	3;IND

As second measure, we provide a tagging mechanism that enables the user to quickly mark predictions with predefined tags. The tags are individually adjustable within the configuration file under the key *markers* and will be displayed within a separate list under the *Info* tab.

3.4.3 Searching and Filtering

A common first task in statistical error analysis is to discover the model results. This often involves atomic processes, such as, searching and filtering which help to reduce analysis space and put the focus on a small subset of the predictions. This way, researchers can detect interesting patterns or outliers more easily which might inform about valuable linguistic characteristics or simply hint to implementation errors. To support researchers in performing these tasks, we added the two concepts of searching and filtering into our analysis tool which are further described in more detail.
For the searching mechanism, we decided to focus on regular expressions since they allow for more complex searching patterns without the need of any sophisticated search engine as introduced within the paper of Gonzales et. al.. Furthermore, this approach seems reasonable since we mainly address people who are related to the fields of computer science and expect them to be familiar with the concepts of regular expressions.
Regarding the filtering mechanism we focused on various linguistic characteristics, such as grammatical and syntactical features. For the grammatical features we focused on the six main grammatical categories: lexis, number, case, mood, tense and person as defined in the Stuttgart-Tübingen-TagSet (STTS)[1]. In addition, we allow users to filter on syntactical features, such as, the number of words within a term or on statistical metrics, such as, the Levenshtein distance.

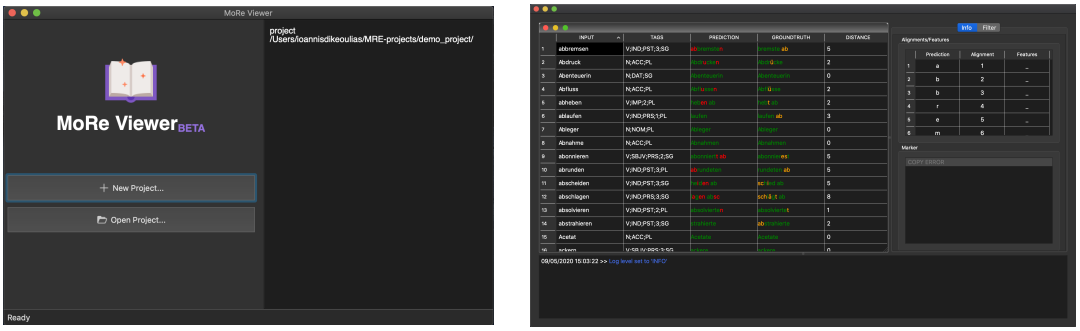
3.5 Graphical User Interface

Graphical User Interfaces (GUIs) have considerable benefits over interactive command-line tools. They provide enhanced visual clues to important system functionalities, enable advanced interaction possibilities and promote a consistent layout and user experience. Nevertheless, a persistent challenge when designing graphical helper tools is to find a good balance between productivity and usability. While it is important to provide useful and powerful tools to the user, it should not be at the expense of the visibility and accessibility of information.
Thereof, we focused on a flexible organisation of subwindows and an intuitive and effortless access to important analysis tools and information views. To provide a better intuition on our design decisions, the next section will introduce the different views and display modes of our error analysis tool and describe their affordances.

3.5.1 Views and Display Modes

In this section we will provide a detailed description of the individual views and display modes of the MoRe Viewer error analysis tool.

[1] https://www.ims.uni-stuttgart.de/forschung/ressourcen/lexika/germantagsets/



(a) The start window.

(b) The main window.

Figure 2: Two screenshots of the graphical error analysis tool MoRe Viewer.

Start Window

The start window serves as starting point for the analysis tool. It includes the application logo and a quick access to the frequent menu actions *New Project* and *Open Project*. In addition, it features a *Recent Projects* widget that lists previously saved projects and allows immediate access without any additional configuration.

Main Window

The main window organises three subwindows, namely the MDI area, the tab widget and the logging area. The subwindows are dynamically resizable which allows to fit the views according to the currently required space. Especially the MDI area which may include multiple documents can thereby flexibly adjusted to the users needs.

MDI Area The MDI (Multiple Document Interface) area allows to import and display multiple documents side-by-side. It further allows to analyse multiple model outputs simultaneously and enables a chronological comparison of different model predictions, while supporting new possibilities for temporal statistics.

MDI Subwindow As mentioned above, the MDI area organises multiple subwindows. These subwindows show filter tables that are populated with the predictions from the model results. According to the specified result schema (cf. 3.4.1), these tables will display the columns *input*, *tags*, *prediction*, *groundtruth* and *distance*. All columns, except the *distance*, which is calculated dynamically, have to be included in the model output provided by the user.

Info Tab The info tab as first part of the tab widget includes additional information about the individual predictions, such as word alignments and character-based feature information. Further, it provides a configurable tagging area, which allows to mark predictions with user-defined tags and a free-text comment section for individual notes. (cf. 3.4.2)

Filter Tab The filter tab as second part of the tab widget includes multiple controls for the error analysis. The controls are individually adjustable for each subdocument within the MDI area. Currently, the following utilities are accessible (cf. 3.4.3):

- A set of selection controls for each individual grammatical feature as defined by the STTS,
- a textual input control expecting a regular expression,
- a numeric input control expecting a Levenshtein distance,
- a numeric input control expecting a word count,
- a selection field offering different statistics (cf. 3.5.3) to be displayed and
- a reset button that resets all controls to their default values.

Screenshots of the *Info* and *Filter* tabs can be found in the appendix B.

Logging Area The main purpose of the logging area is to provide a feedback mechanism for the user, which includes basic information about the current system state, e.g. number of filter results and debugging information. Currently, the following logging levels are supported:

- OFF: no logging is performed.
- DEBUG: fine grained debugging information, e.g. system prints.
- INFO: coarse-grained progress information, e.g. system state.
- WARNING: notices about possible harmful events.
- ERROR: errors that will lead to undefined behaviour.
- CRITICAL: severe errors that will lead the system to quit.
- ALL: all levels are active.

All log levels, except *OFF* and *ALL* show only the information for the currently active log level, e.g., *DEBUG* shows only debug information.
For future versions of this tool, we aim to improve the logging area since it can provide a powerful extension for users to interact with their model results. For example, we could integrate shortcuts that allow more advanced users to build semi-automatic workflows

that perform different actions in sequence. Such a transaction could use a SQL-like syntax as shown below:

```
BEGIN ACTIONS
SELECT prediction FROM results WHERE editDistance > 5 AS alias1;
TAG alias1 AS edit-larger-5
END
```

The workflow as shown above would select all predictions from the result set that have an edit distance greater five and tag them with the name *edit-larger-5*.

3.5.2 Error Types and Highlighting

In linguistics, there exist mainly two types of errors: Interlingual errors describe errors that are caused by the wrong utilisation of rules and patterns between two languages. For instance, in German, a common ending for *nouns* in *plural er*, which is not the case in English.
Intralingual errors on the other hand describe errors that occur within the scope of a single language, often caused by simplification or overgeneralisation and usually a result of data sparsity. Our tool does not distinguish between error types neither does it specify which class an error belongs to. We solely focus on the highlighting of words which is performed in two opposite but symmetric directions:

Prediction colouring Prediction colouring highlights all characters of the prediction term in *red* which are not part of the groundtruth term.

Groundtruth colouring Groundtruth colouring highlights all characters of the groundtruth term in *orange* which are not part of the prediction term.

3.5.3 Statistics and Evaluation Metrics

To provide researchers a statistical foundation on which they can base their hypotheses and decisions, we provide two different statistical measures. First, as part of the table views (cf. 3.5.1) we provide the popular *Levenshtein distance*, also known as *edit distance*. This metric computes the number of basic operations (insert, delete, substitute) that are required to transform an input word into a target word. Within our scope, we choose the *prediction* as *input word* and the *groundtruth* as *target word*.
Secondly, we offer simple descriptive statistics for the result set to provide a quick first impression on the result set. Currently, our tool provides the following quantitative statistics:

- number of rows per edit distance,
- number of rows per grammatical tag,
- median distance per grammatical tag and
- mean distance per grammatical tag.

Each statistic can be individually displayed as a graphical plot and opens within a modal window for better visibility.

4 Discussion and Limitations

Error analysis tools are researchers Swiss army knife for the evaluation of statistical NLP models. Nevertheless, many tools lack a graphical user interface and even worse a good user experience or even appropriate usability measures in order to be used by a broader audience. Furthermore, these tools are often tailored to specific usage scenarios which do not allow for any abstraction regarding the model architecture or result schema. Additionally, researchers usually have different expectations on their statistical models which often lead to conflicting requirements for the design of error analysis tools.
Useful approaches would incorporate broad field studies that could offer valuable results with a high ecological validity. However, these studies are usually time-consuming and expensive. Additional challenges to these problems are often given by the high complexity of natural language processing tasks and the related efforts and expenses. Moreover, many researchers struggle to contribute to long studies which do not incorporate their own research interests.
This work extended a research study that occupied with the development of a statistical model for the task of morphological reinflection. Thereby, we gained many insightful ideas on how to design and develop an error analysis tool that can be used by computational linguists as well as research assistants. Although, we steadily focused to build a tool that is usable in broad research scopes, we need to restrict ourselves to specific requirements in order to meet time restrictions and allow for a fast realisation of this tool. Still, MoReViewer is build with the view to provide a valuable extension to researchers tool kits and allow for interesting usage cases in future tasks.

5 Conclusion and Future Work

In this paper we presented MoReViewer, a graphical error analysis tool that supports computational linguists and research assistants in common evaluation tasks during the development of morphosyntactic learning models. Therefore, we designed and developed a simple and easy to use graphical user interface that features many helpful functionalities for error analysis, such as, searching for specific word patterns, filtering on syntactical and grammatical features or sorting on linguistic evaluation metrics. Further, our tool

provides various simple and extensible statistics that allow researchers to get first impressions on the results via integrated plots.
In addition, we integrated useful evaluation metrics, such as, the Levenshtein distance to facilitate and accelerate the detection of critical errors. Furthermore, we were particularly interested in designing a tool that provides a contemporary and modern design which fluidly adapts to researchers development environments and provides a lightweight and enjoyable extension to their everyday tasks. Thereof, we put a special focus on various HCI guidelines and best practices, such as, aesthetic and minimalistic design of views, various error prevention and recovering techniques as well as a consistent layout of input controls to provide an enhanced experience for our users. In addition, our tool promotes a flexible language schema that allows researchers to easily specify their individual grammar within a simple JSON configuration file, while enabling the analysis of model predictions from any language or grammar.
Regarding future work, we aim to integrate various information retrieval mechanisms that should help to automatically identify errors within model predictions and suggest possible improvements based on integrated evaluation metrics. Possible approaches would incorporate lexical databases, such as WordNet or EuroNet that offer powerful libraries and APIs for information retrieval. Beyond that, it would be of great interest to investigate to what extent semi-structured information from online libraries, such as, Wiktionary could be used in order to improve the automation of error analysis for NLP tasks. In addition, computer-aided error analysis (CEA) opens a broad research field with many challenges and interesting research topics.
To this end, possible research studies could investigate whether morphological statistics incorporating syntactical and phonetical features significantly correlate with the error rate. Further, this tool would greatly benefit from a more fine-grained classification of error types in order to provide useful suggestions for error correction. Furthermore, researchers could perform a user study with computational linguistics to evaluate the effectiveness and satisfaction of error analysis tools, which could give us insightful results on necessary requirements and improvements.

6 License and Contact Information

The error analysis tool is available[2] for free for research purposes under the GNU General Public License as published by the Free Software Foundation.

[2] https://github.com/JohnyDevv/MoReViewer

Acknowledgments

I want to thank Prof. Dr. Dietrich Klakow for his encouragement and the opportunity to research and develop freely and creatively in a comfortable atmosphere. Further, I want to thank all course members for the interesting and joyful discussions during our meetings. My deep gratefulness and thanks go to Sasha Mayn and Matthias Lindemann for their continuous support and their insightful comments and ideas which greatly contributed to the development of this tool.

References

[1] Apoorv Agarwal, Ankit Agarwal, and Deepak Mittal. An error analysis tool for natural language processing and applied machine learning. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: System Demonstrations*, pages 1–5, 2014.

[2] Ahmed El Kholy and Nizar Habash. Automatic error analysis for morphologically rich languages. *Proc. of the MT Summit XIII, Xiamen, China*, pages 225–232, 2011.

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

[4] Meritxell Gonzalez, Jesús Giménez, and Lluís Màrquez. A graphical interface for mt evaluation and error analysis. In *Proceedings of the ACL 2012 System Demonstrations*, pages 139–144. Association for Computational Linguistics, 2012.

[5] Lung-Hao Lee, Liang-Chih Yu, and Li-Ping Chang. Overview of the nlp-tea 2015 shared task for chinese grammatical error diagnosis. In *Proceedings of the 2nd Workshop on Natural Language Processing Techniques for Educational Applications*, pages 1–6, 2015.

[6] Elijah Mayfield and Carolyn Rosé. An interactive tool for supporting error analysis for text mining. In *Proceedings of the NAACL HLT 2010 Demonstration Session*, pages 25–28, 2010.

[7] Maja Popović, Hermann Ney, Adrià De Gispert, José B Mariño, Deepa Gupta, Marcello Federico, Patrik Lambert, and Rafael Banchs. Morpho-syntactic information for automatic error analysis of statistical machine translation output. In *Proceedings of the workshop on statistical machine translation*, pages 1–6. Association for Computational Linguistics, 2006.

[8] Shijie Wu and Ryan Cotterell. Exact hard monotonic attention for character-level transduction. *arXiv preprint arXiv:1905.06319*, 2019.

Appendices

A Configuration Files

Below, two sample configuration files for the result and grammar schema are shown.

Result schema

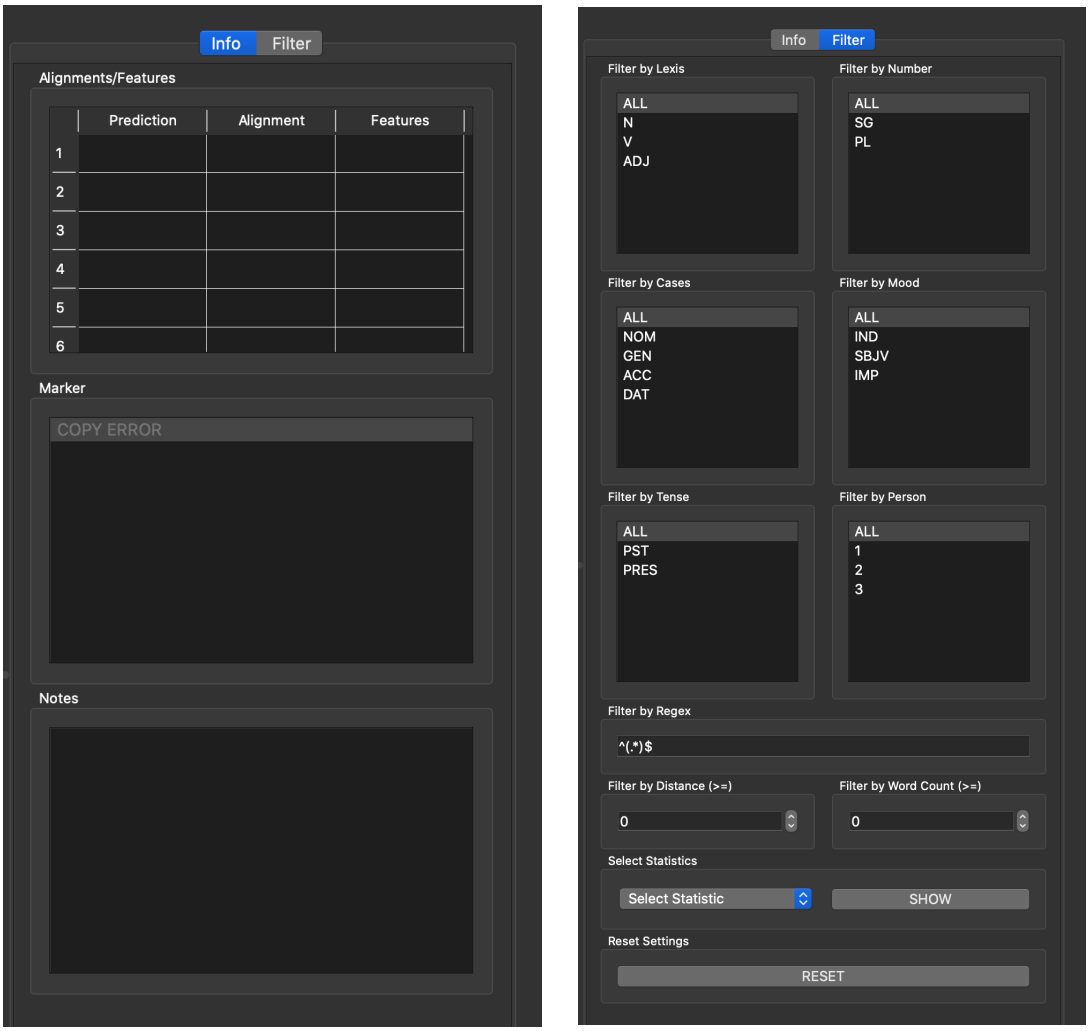
```
{
    "columns": {
        "input": 1,
        "prediction": 2,
        "tags": 3,
        "ground-truth": 4,
        "alignment": 0,
        "features": 0
    },
    "markers": {
        "COPY ERROR",
        "FEATURE ERROR",
        "DATA SPARSITY"
    }
}
```

Grammar schema

```
{
    "grammar":{
        "LEXIS":{
            "N": "Noun",
            "V": "Verb",
            "ADJ": "Adjective"
        },
        "NUMBER":{
            "SG": "Singular",
            "PL": "Plural"
        },
        "CASES":{
            "NOM": "Nominative",
```

```
            "GEN": "Genitive",
            "ACC": "Accusative",
            "DAT": "Dative"
        },
        "MOOD":{
            "IND": "Indicative",
            "SBJV": "Subjunctive",
            "IMP": "Imperative"
        },
        "TENSE":{
            "PST": "Past",
            "PRES": "Present"
        },
        "PERSON":{
            "1": "First",
            "2": "Second",
            "3": "Third"
        }
    },
    "grammar-dependencies": {
        "LEXIS:N": [
            "NUMBER",
            "CASES"
        ],
        "LEXIS:V": [
            "NUMBER",
            "MOOD",
            "TENSE",
            "PERSON"
        ]
    }
}
```

B Info and Filter Tabs



(a) The info tab. (b) The filter tab.

Figure 3: The two tabs of the control widget.