A Web-based Annotation Framework for Large-scale Text Correction

Ossama Obeid[1] **Wajdi Zaghouani**[1] **Behrang Mohit**[1]
Nizar Habash[2] **Kemal Oflazer**[1] **Nadi Tomeh**[2]

[1]Carnegie Mellon University in Qatar
{oobeid@,wajdiz@,behrang@,ko@cs.}cmu.edu

[2]Center for Computational Learning Systems, Columbia University
{habash,nadi}@ccls.columbia.edu

Abstract

We demonstrate a web-based, language-independent annotation framework used for manual correction of a large Arabic corpus. Our framework provides intuitive interfaces for annotating text and managing the annotation process. We describe the details of both the annotation and the administration interfaces as well as the back-end engine. We also show how this framework is able to speed up the annotation process by employing automated annotators to fix basic Arabic spelling errors.

1 Introduction

Errors in natural language text, such as incorrect spelling, word choice, or grammar, are problematic for natural language processing (NLP) systems: they contribute to data sparseness and limit the effectiveness of NLP models. Automatic correction of these errors have been studied for different languages (Kukich, 1992). QALB (Qatar Arabic Language Bank)[1] is a project on automatic correction of errors in Arabic text. Our approach has two components: (a) large scale manual annotation (correction) of Arabic errors, and (b) statistical modeling of the text correction.

In this paper we focus on the first task and describe the design and implementation of our language-independent, web-based annotation system. Our framework provides intuitive interfaces for both managing the annotation process and performing the annotations. Additionally, we show how our framework employs automatic annotators to correct basic Arabic spelling mistakes to speed up the annotation process.

Our framework consists of two major interfaces: (a) an Admin interface, which enables the lead annotator to create, assign, monitor, evaluate and export annotation tasks in large scale; and (b) an Annotation interface, which enables annotators to conduct and review annotation tasks for different types of text. The interface is flexible for handling monolingual annotation (e.g. Arabic text) and also bilingual annotation (e.g. post-editing of MT output). The interface provides the required annotation functionality for moving, merging, replacing and editing words within a paragraph along with the undo and redo actions.

In addition to the Admin tools, the framework provides the lead annotator with components for automatic correction of basic errors and also quality control. The annotation framework is currently deployed and is expected to be used by up to twenty users, annotating an aggregated corpus of two million words.

[1]http://nlp.qatar.cmu.edu/qalb

2 Related Work

Traditionally, manual text correction is performed under the context of post-editing machine translation (MT) output. The goal of post-editing is to evaluate MT systems rather than building corpora of edits.

Tools like PET (Aziz et al., 2012) and BLAST (Stymne, 2011) provide annotators a text-editor-like interface to identify, record, and correct errors. Text-editor-like interfaces are very flexible and allow all forms of corrections to be performed, but, they are not capable of accurately tracking token alignment, if at all.

Frameworks such as EXMARaLDA (Schmidt, 2010) and GATE (Cunningham et al., 2011) facilitate multi-layer and multi-round annotations. An example of such approach is the work of Dickinson and Ledbetter (2012) who annotated errors in Hungarian students essays using multiple annotation layers from phonology to syntax in different stages.

TCTool (Llitjós and Carbonell, 2004) provides a token-based correction interface. It allows for tokens to be moved around, deleted, or added, but, does not allow for tokens to be merged or split.

This is because it assumes all input text to be outputs of MT systems, which do not produce merged or split words. Since our source text contains a majority of manually written text, this assumption does not hold. Furthermore, TCTool is designed to deal with short sentences while we aim to annotate larger documents in order to benefit from wider context.

Above all, these tools are not designed for large-scale, distributed annotation projects. They do not provide facilities for managing thousands of documents, distributing tasks to tens of annotators and evaluating inter-annotator agreement (IAA). Our system draws on the advantages of the above tools while adding the required facilities to manage a large annotation project. In this aspect, our system is similar to the COLABA project annotation tool, a web application for dialectal Arabic text annotation (Benajiba and Diab, 2010; Diab et al., 2010).

3 System Requirements and Constraints

The QALB project aims to produce a large corpus (two million words) of manually corrected Arabic text. Our system must allow for quick annotation of texts without sacrificing annotation correctness and consistency. This section describes the requirements and constraints that our annotation system needs to fulfill.

Text Correction The QALB corpus will contain corrections of errors produced by native speakers of various dialects, non-native speakers and machine translation systems in a variety of contexts including news, Wikipedia articles, forum posts, and student essays. Therefore, our annotation system should not just account for spelling mistakes. Our annotation interface allows annotators to perform different types of *actions* which correspond to the following types of corrections: (a) *Edit* actions: words that are misspelled or mistyped should be modified. (b) *Move* actions: Words that are not in the right location should be moved to the right location. (c) *Add* actions: words that are missing need to be added. (d) *Delete* actions: extraneous words should be deleted. (e) *Merge* actions: words that have been split by mistake should be merged. (f) *Split* actions: words that have been merged by mistake should be split.

Token Alignment Since we allow a large range of corrections, we need to be able to track the alignment of corrected tokens to the original text. In addition to token alignment, we want to track the list of actions performed by each annotator in the hope that we may learn from the human correction process.

Efficiency Due to the large amount of text that needs to be corrected, our Annotation interface should allow annotators to concurrently log into the system and perform their annotation tasks very quickly.

Quality Control To ensure the quality of corrections in our corpus we should be able to monitor the performance of each individual annotator and the consistency of annotators among each other. Therefore, our system should allow us to perform inter-annotator agreement (IAA) evaluation. We also need to ensure that all source documents are of reasonable quality. For this we need a mechanism for annotators to flag low quality (e.g. highly dialectal) text.

4 Annotation Web Interface

In this section we present our framework, the Annotation Web Interface, which will be used to carry out the annotation process. Our main contribution is the Annotation interface (Figure 1) which provides an intuitive drag-and-drop interface to manipulate tokens in a document. We describe the design and implementation of each component in further details.

4.1 Architecture

Our framework has three core components: the Annotation interface, the Admin interface, and the application programming interface (API) server.

The Annotation interface is used by annotators to correct assigned documents. The Admin interface is used by the lead annotator to manage annotators and documents, assign tasks, evaluate IAA, and monitor the overall progress of the annotation process. Figure 2 illustrates how these components interact with each other.

Both Admin and Annotation interfaces are web pages that complete their respective tasks by sending HTTP requests to the API server. The API server handles these requests and performs the necessary operations using the local file system and a database.

In addition to the three core components, there are automated annotators. Automated annotators are scripts that interface with the API server to perform automatic corrections. We discuss how we use an automated annotator to speed up the annotation process.

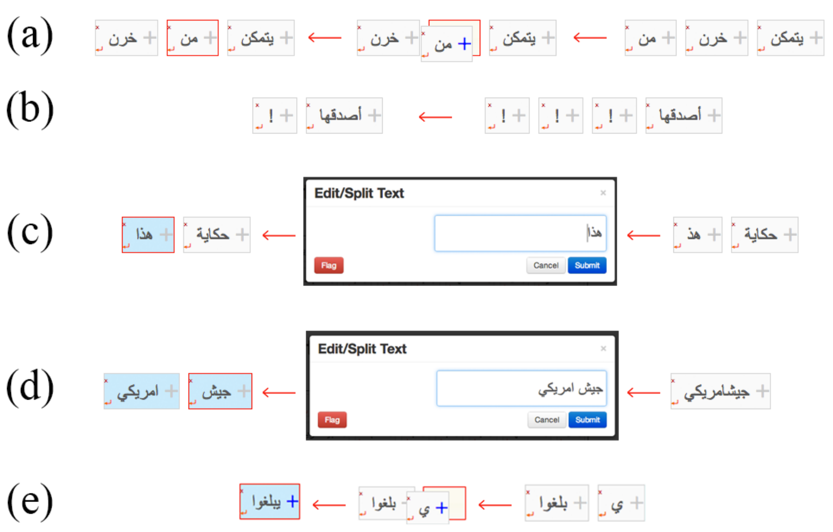

Figure 1: Sample of corrections of a token: (a) Moving (b) Deleting (c) Editing (d) Splitting (e) Merging

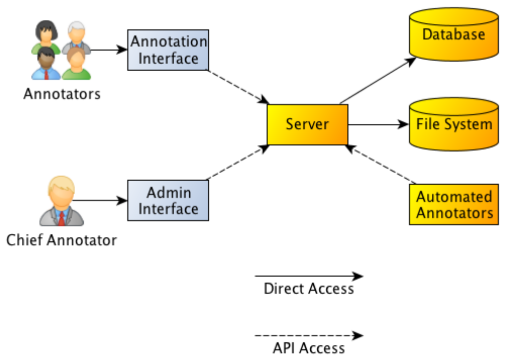

Figure 2: General Architecture Diagram.

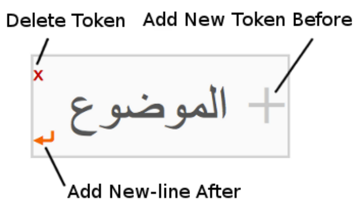

Figure 3: A single token box.

4.2 Annotation Interface

Annotators need to be able to view their assigned tasks, perform corrections, and submit their final corrections. This is done through the Annotation Interface. The Annotation Interface first displays a list of tasks assigned to an annotator. The annotator selects a task and then displays the Annotation Window, where corrections are performed.

The Annotation Window displays tokens in separate boxes. Each box can either be dragged or double-clicked. Tokens can be moved by dragging and dropping tokens to the desired location. Tokens can be merged by dragging a token slightly to the left or to the right to be merged with the previous or the next token, respectively. Double clicking on a token opens a dialog box with a text input which contains the current value of the clicked token and can be used to modify the token's text. Adding a space between two characters of a token performs a split. Annotators cannot modify a token and split it at the same time. This allows us to track individual changes so that we have a consistent action history.

As illustrated in Figure 3, each box has additional buttons that can be used to either delete a token, add a new token before the selected token, or add a new line after the selected token. Figure 1 shows screen-shots of each action performed in sequence.

The Annotation Interface also has few other features to help with the annotation task. *Undo* and *Redo* buttons are provided to allow annotators to go back and fix mistakes. The interface also provides access to both the original Arabic text, and, if the text was the output of a machine translation system, the original English text. This additional information help annotators in determining how much their corrections alter the meaning of the original text. If a document has poor quality of writing or translation, the interface provides the annotator with a *Flag* button, which alerts the lead annotator about the issue.

4.3 Admin Interface

The lead annotator will be managing a team of about twenty annotators who can use the system remotely and concurrently. The Admin Interface contains: (a) a user management tool for creating new annotator accounts and viewing annotator progress; (b) a document management tool for uploading new documents, assigning them for annotation, and viewing submitted annotations; and (c) a monitoring tool for viewing overall annotation progress and evaluating IAA.

4.4 API Server and Automated Annotators

The API server lies at the heart of our framework. It is a Python server that provides a web API through HTTP requests for retrieving, creating, and modifying content such as user records, source documents, and annotation submissions. All responses by the API server are JSON objects. This allows us to easily create dynamic web pages for the Annotation and Admin interfaces as well as automated annotators. One automated annotator we deployed to automatically correct ة (*Ta Marbuta*) versus ه (*Ha*) errors and ء (*Hamza*) placement errors by running each document through the MADA system (Habash et al., 2009). All documents are corrected by our automated annotator before being assigned to annotators to cut down annotation time.

5 Demonstration Script

During the demonstration, we will present the use of the Admin and Annotator interfaces using simple and complex examples of various kinds of edits as discussed above. In particular, we will show how the Annotation tool can be used for correcting a sample piece of text using the various allowed operations.

6 Conclusion and Future Work

We presented a detailed overview of our web-based annotation framework for correcting writing errors. Deployment for error correction in other languages is a natural extension of this work since almost all functionalities of our system are language independent. In the future, we plan to include new functionalities for increasing annotators' search and lookup power and a web-based component for training new annotators. We also plan to make use of the created annotations to develop automatic error detection and correction systems.

7 Acknowledgements

We thank anonymous reviewers for their valuable comments and suggestions. This publication was made possible by grants NPRP-4-1058-1-168 and YSREP-1-018-1-004 from the Qatar National Research Fund (a member of the Qatar Foundation). The statements made herein are solely the responsibility of the authors.

References

Wilker Aziz, Sheila Castilho Monteiro de Sousa, and Lucia Specia. 2012. PET: a tool for post-editing and assessing machine translation. In *Proceedings of the LREC'2012*.

Yassine Benajiba and Mona Diab. 2010. A web application for dialectal Arabic text annotation. *Proceedings of the LREC Workshop for Language Resources (LRs) and Human Language Technologies (HLT) for Semitic Languages: Status, Updates, and Prospects*.

Hamish Cunningham, Diana Maynard, Kalina Bontcheva, Valentin Tablan, Niraj Aswani, Ian Roberts, Genevieve Gorrell, Adam Funk, Angus Roberts, Danica Damljanovic, Thomas Heitz, Mark A. Greenwood, Horacio Saggion, Johann Petrak, Yaoyong Li, and Wim Peters. 2011. *Text Processing with GATE (Version 6)*. University of Sheffield.

Mona Diab, Nizar Habash, Owen Rambow, Mohamed Altantawy, and Yassine Benajiba. 2010. Colaba: Arabic dialect annotation and processing. *LREC Workshop on Semitic Language Processing*, pages 66–74.

Markus Dickinson and Scott Ledbetter. 2012. Annotating errors in a Hungarian learner corpus. In *Proceedings of the LREC'2012*.

Nizar Habash, Owen Rambow, and Ryan Roth. 2009. MADA+TOKAN: A toolkit for Arabic tokenization, diacritization, morphological disambiguation, pos tagging, stemming and lemmatization. In *Proceedings of the Second International Conference on Arabic Language Resources and Tools*.

Karen Kukich. 1992. Techniques for automatically correcting words in text. *ACM Comput. Surv.*, 24(4):377–439, December.

Ariadna Font Llitjós and Jaime G. Carbonell. 2004. The translation correction tool: English-Spanish user studies. In *Prceedings of the LREC'04*.

Thomas Schmidt. 2010. Linguistic tool development between community practices and technology standards. In *Proceedings of the LREC Workshop Language Resource and Language Technology Standards*.

Sara Stymne. 2011. Blast: a tool for error analysis of machine translation output. In *Proceedings of the ACL'2011: Systems Demonstrations*, pages 56–61.