DKPro-UGD: A Flexible Data-Cleansing Approach to Processing User-Generated Discourse

Richard Eckart de Castilho and Iryna Gurevych

Ubiquitous Knowledge Processing Lab
Technische Universität Darmstadt, Germany
http://www.ukp.tu-darmstadt.de

Abstract. User-generated discourse[1] from Web 2.0 poses particular challenges to natural language processing (NLP) due to its noise and error proneness. A data cleansing step preceding the analysis steps in an NLP pipeline can reduce the problems. While recent efforts provide general-purpose collections of UIMA-based analysis components, data cleansing seems not yet to be covered. The five-stage data cleansing approach proposed here offers a maximum of flexibility in identifying problematic artifacts, deciding how to deal with them and analysing cleansed data. Simultaneously, it allowed us to create reusable UIMA-based components for the actual data cleansing and for mapping annotations created on the clean data back to the original representation. These components are released as part of the Darmstadt Knowledge Processing Software Repository (DKPro) under the name of *DKPro-UGD*.

1 Introduction

In recent years, the World Wide Web has seen a rapid growth of user-generated discourse like wikis, blogs, chats, social question & answer platforms, etc. (Web 2.0). User-generated text from such sources is often polluted by spelling errors, shorthand or other artifacts that existing natural language processing tools have difficulty dealing with. Cleansing the data by removing or substituting such artifacts can reduce or eliminate these difficulties.

UIMA [1] is rapidly being adopted by the language processing community and used in many NLP projects as an integration framework to build processing pipelines from diverse analysis components that traditionally have been hard to combine. We know of several institutions that provide repositories of ready-to-use UIMA components: the Ubiquitous Knowledge Processing (UKP) Lab at Technische Universität Darmstadt, Germany provides the Darmstadt Knowledge Processing Software Repository (DKPro) [2]; JULIE Lab at Friedrich-Schiller-Universität, Jena, Germany provides an UIMA-based NLP tool suite

[1] We define *user-generated discourse* as text or dialogue in written form created by ordinary web users without any editorial control.

[3]; Carnegie Mellon University, Pittsburgh, PA, USA hosts an UIMA component repository [4]. To our knowledge, data cleansing is not yet addressed by components in these repositories.

In this contribution, we present a flexible approach for handling data cleansing as part of a regular UIMA pipeline. The approach involves generic UIMA components that can be adapted to local usage scenarios via configuration files. An important feature of the approach is the transfer of annotations created on the cleansed data back to the original data. This allows to append sophisticated analysis steps towards the end of the pipeline to take annotations on the cleansed data into account as well as the original unmodified representation.

This contribution is created in the context of the DKPro[2] project. The DKPro repository consists of several main parts serving the purposes of different NLP application areas. *DKPro-Core* provides general purpose analysis components for tasks such as segmentation, POS-tagging, parsing, etc. *DKPro-IR* supplies components for all phases of information retrieval, including indexing, retrieval and evaluation. *DKPro for Text Mining* includes readers for popular Web 2.0 sites and analysis components for opinion- or sentiment-related properties of the often highly subjective content provided on these sites. The presented components are part of *DKPro-UGD*, which provides components for preprocessing user-generated discourse and analysis components for the identification and handling of errors and artifacts (emoticons, shorthand, etc.) commonly present in user-generated discourse.

2 Sources of noise in user-generated discourse

Common sources of noise in user-generated discourse include spelling errors, typos, shorthand, and grammatical mistakes. In addition, the use of artifacts such as emoticons or even basic markup is particularly popular in forums or chat rooms. Depending on the source of noise, different actions may be required depending on the usage scenario. Thus, a general solution needs to provide a maximum of flexibility in identifying noise and deciding how to deal with it.

Pattern matching may be employed to identify artifacts like emoticons or markup. A typical decision is to remove those artifacts from the text before applying NLP tools to it. However, those artifacts may be important in certain use cases. In sentiment analysis, for example, an emotionally connotated (e.g. happy or sad) smiley is a valuable piece of information. Therefore, it is important that the artifact is hidden only from those NLP tools for which it poses problem.

Shorthand or abbreviations may be expanded to full phrases by means of dictionaries. For example, the sentence *But of course IANAL* usually poses a problem for NLP tools. The expanded sentence *But of course I am not a lawyer*, however, is not problematic at all. Expansion is not a general solution for all shorthand though. For example, *lol* (laughing out loud) or *roftl* (rolling on the floor laughing) seem to be used more like emoticons than like abbreviations and thus expanding them may actually break an otherwise correct sentence.

[2] DKPro website: http://www.ukp.tu-darmstadt.de/software/dkpro/

Various methods of detecting spelling errors and typos exist. The error has to be corrected in order to remove the resulting problem for subsequently applied NLP tools. For correcting the error, again, a number of approaches exist. Therefore, it is essential that in order to allow for maximal flexibility, the data cleansing process makes a distinction between detecting an error and deciding how to deal with it.

3 Proposed Approach

While in general the data cleansing process has to be adapted to the particular context of use, we aim at producing reusable processing components. To address the trade-off between generality and specificity that exists here we define five principal processing stages. Section 4 will then present concrete implementations for the two core stages *apply* and *undo*.

Stage 1: Identify – Problematic artifacts are identified and annotated. A simple realization of this stage may employ a method which does not depend on previous analysis steps, e.g. a pattern- or dictionary-based method. A more sophisticated realization of this stage may involve multiple analysis steps, e.g. tokenisation followed by spelling error detection.

Stage 2: Decide – A decision is made how to deal with the problematic artifacts. For example, the decision for a particular correction of a spelling error is made. This decision is then encoded as an *edit* annotation identifying the edit operation to be applied to the annotated location in the next stage. Possible edit operations are *insert*, *delete* and *replace*. In some cases, this stage may be conflated with the previous stage.

Stage 3: Apply – All *edit* operations are applied to the data. The resulting data is stored in a new version (called *view* in UIMA) without changing the original. Modifications are tracked in such a way that it is possible to find the corresponding original data for an arbitrary section of the modified data.

Stage 4: Analyse – The cleansed version is processed by regular analysis components. It is also possible to insert a nested cleansing process at this stage. For example, the first outer cleansing process may remove markup, while a second nested cleansing process may address spelling correction.

Stage 5: Undo – In this phase, all edit operations are effectively reverted. This happens by copying all annotations created during the *Analyse* stage over to the original data and subsequently adjusting the stand-off anchors using the tracking information recorded in the *Apply* stage.

4 Implementation

In our implementation of the process outlined above, the UIMA components (*ApplyChangesAnnotator* and *Backmapper*) realize the *Apply* and *Undo* stages. We defined the UIMA type (*SofaChangeAnnotation*) to encode the *edit* operations. The realization of the *Identify*, *Decide* and *Analyse* stages is intentionally left open to ensure maximal flexibility. During development and testing, we

used the UIMA Sandbox *RegularExpressionAnnotator* and the *SpellChecker* and *SpellingCorrector* components from DKPro in the *Identify* and *Decide* stages as well as a tokenizer, sentence splitter and part-of-speech tagger from the DKPro component repository in the *Analyse* stage.

The *SofaChangeAnnotation* used to encode the *edit* operations bears three features. The feature *operation* takes one of the values *insert*, *delete* or *replace* and defines the edit operation to take place. The *insert* and *replace* operations require that the feature *value* is filled.

In the *Apply* stage, the *ApplyChangesAnnotator* applies all the changes encoded by *SofaChangeAnnotation*s to the text in the *source view*, thereby generating a new *target view* containing the modified text and no annotations. In case of overlapping edits, only the left-most and longest edit operation is executed.

In the *Undo* stage, the *Backmapper* copies all annotations from its *source view* to the *target view*. The mapping for these two views has to be exactly reversed from the mapping specified for the *ApplyChangesAnnotator*. After copying the annotations, the *AlignedString* data structure is fetched and used to adjust the *begin* and *end* features of all copied annotations to the uncleansed data.

5 Conclusion

User-generated discourse often contains noise that poses significant problems to existing NLP tools. A data cleansing step can reduce or eliminate these problems. We have presented a data cleansing approach and its implementation that employs the flexibility of UIMA to allow for adaptation to particular usage scenarios based on reusable components. Mapping annotations on the cleansed data back to the original representation allows for comprehensive analysis. The produced components are provided as part of the DKPro-UGD 1.0 release.

Acknowledgements: The Darmstadt Knowledge Processing Software Repository is a joint project of all members of the UKP Lab. DKPro is partially supported by IBM Unstructured Information Analytics Innovation Awards given to the UKP Lab in 2007 and 2008.

References

[1] Ferrucci, D., Lally, A.: UIMA: an architectural approach to unstructured information processing in the corporate research environment. Natural Language Engineering **10**(3-4) (2004) 327–348

[2] Müller, C., Zesch, T., Müller, M.C., Bernhard, D., Ignatova, K., Gurevych, I., Mühlhäuser, M.: Flexible UIMA components for information retrieval research. In: Proceedings of the Workshop Towards Enhanced Interoperability for Large HLT Systems: UIMA for NLP, Marrakech, Morocco, LREC (May 2008)

[3] Hahn, U., Buyko, E., Tomanek, K., Piao, S., McNaught, J., Tsuruoka, Y., Ananiadou, S.: An annotation type system for a data-driven NLP pipeline. In: Proceedings of the Linguistic Annotation Workshop, Prague, Czech Republic, Association for Computational Linguistics (June 2007) 33–40

[4] Carnegie Mellon University: UIMA component repository. http://uima.lti.cs.cmu.edu:8080/UCR/