jMWE: A Java Toolkit for Detecting Multi-Word Expressions

Nidhi Kulkarni & Mark Alan Finlayson
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, MA, 02139, USA
{nidhik,markaf}@mit.edu

Abstract

jMWE is a Java library for implementing and testing algorithms that detect Multi-Word Expression (MWE) tokens in text. It provides (1) a detector API, including implementations of several detectors, (2) facilities for constructing indices of MWE types that may be used by the detectors, and (3) a testing framework for measuring the performance of a MWE detector. The software is available for free download.

jMWE is a Java library for constructing and testing Multi-Word Expression (MWE) token detectors. The original goal of the library was to detect tokens (instances) of MWE types in a token stream, given a list of types such as those that can be extracted from an electronic dictionary such as WordNet (Fellbaum, 1998). The purpose of the library is not to discover new MWE types, but rather find instances of a set of given types in a given text. The library also supports MWE detectors that are not list-based.

The functionality of the library is basic, but it is a necessary foundation for any system that wishes to use MWEs in later stages of language processing. It is a natural complement to software for discovering MWE types, such as `mwetoolkit` (Ramisch et al., 2010) or the NSP package (Banerjee and Pedersen, 2003). jMWE is available online for free download (Finlayson and Kulkarni, 2011a).

1 Library Facilities

Detector API The core of the library is the detector API. The library defines a detector interface which provides a single method for detecting MWE tokens in a list of individual tokens; anyone interested in taking advantage of jMWE's testing infrastructure or writing their own MWE token detection algorithm need only implement this interface. jMWE provides several baseline MWE token detection strategies. Also provided are detector filters, which apply a specific constraint to, or resolve conflicts in, the output another detector.

MWE Index jMWE also provides classes for constructing, storing, and accessing indices of valid MWE types. An MWE index allows an algorithm to retrieve a list of MWE types given a single word token and part of speech. The index also lists how frequently, in a particular concordance, a set of tokens appears as a particular MWE type rather than as independent words. To facilitate construction of indices, jMWE provides bindings to the MIT Java Wordnet Interface (JWI) (Finlayson, 2008b) and JSemcor (Finlayson, 2008a), as well as classes which extract all MWE types from those resources and write them to disk.

Test Harness The linchpin of jMWE's testing infrastructure is a test harness that runs an MWE detector over a given corpus and measures its precision and recall. The library comes with default bindings for running detectors over the Semcor corpus or any other corpus that can be mounted with the JSemcor library. Nevertheless, jMWE is not restricted to running tests over Semcor, or even restricted to using JSemcor for interfacing with a corpus: a detector can be run over any corpus whose MWE instances have been marked can be analyzed, merely by implementing four interfaces. Also included in the testing infrastructure are a number of error detectors, which analyze the detailed output of the test harness to identify common MWE token detection errors. The library includes implementation for twelve standard error types.

2 Detection Algorithms

Preprocessing To run an MWE detector over a text the text must, at a minimum, be tokenized. jMWE does not include facilities to do this; tokenization must be done via an external library. Most detection strategies also require tokens to be tagged with a part of speech and lemmatized. This information is also not provided directly by jMWE, but there are bindings in the library for using JWI and the Stanford POS Tagger (Toutanova et al., 2003) to tag and lemmatize a set of texts, provided those texts can be accessed via the JSemcor library.

2.1 Detector Types

MWE token Detectors can be split into at least three types: *Basic Detectors*, *Filters*, and *Resolvers*. Performance of selected combinations of these detectors are given in Table 1.

Basic Detectors that fall into this category use an MWE index, or other source of information, to detect MWE tokens in a stream of tokens. jMWE includes several implementations of basic detectors, including the following:
(1) *Exhaustive:* Given a MWE type index, finds all possible MWE tokens regardless of inflection, order, or continuity.
(2) *Consecutive:* Given a MWE type index, finds all MWE tokens whose constituent tokens occur without other tokens interspersed.
(3) *Simple Proper Noun:* Finds all continuous sequences of proper noun tokens, and marks them as proper noun MWE tokens.

Filters These MWE detectors apply a particular filter to the output of another, wrapped, detector. Only MWE tokens from the wrapped detector that pass the filter are returned. Examples of implemented filters are:
(1) *In Order:* Only returns MWE tokens whose constituent tokens are in the same order as the constituents listed in the MWE type's definition.
(2) *No Inflection:* Removes inflected MWE tokens.
(3) *Observed Inflection:* Returns base form MWEs, as well as those whose inflection has been observed in a specified concordance.
(4) *Pattern Inflection:* Only return MWE tokens whose inflection matches a pre-defined set of part of speech patterns. We used the same rules as those found in (Arranz et al., 2005) with two additional rules related to Verb-Particle MWEs.

Resolvers Like filters, these wrap another MWE detector; they resolve conflicts between identified MWE tokens. A conflict occurs when two identified MWE tokens share a constituent. Examples include:
(1) *Longest-Match-Left-to-Right:* For a set of conflicting MWE tokens, picks the one that starts earliest. If all of the conflicting MWE tokens start at the same point, picks the longest.
(2) *Observed Probability:* For a set of conflicting MWE tokens, picks the one whose constituents have most often been observed occurring as an MWE token rather than as isolated words.
(3) *Variance Minimizing:* For a set of conflicting MWE tokens, picks the MWE token with the fewest interstitial spaces.

Detector	**F_1 (precision/recall)**
Exhaustive +Proper Nouns	0.197_{F_1} (0.110_p/0.919_r)
Consecutive +Proper Nouns	0.631_{F_1} (0.472_p/0.950_r)
Consecutive +Proper Nouns +No Inflection +Longest-Match-L-to-R	0.593_{F_1} (0.499_p/0.731_r)
Consecutive +Proper Nouns +Pattern Inflection +More Frequent As MWE	0.834_{F_1} (0.835_p/0.832_r)

Table 1: F-measures for select detectors, run over Semcor 1.6 brown1 and brown2 concordances using MWEs drawn from WordNet 1.6. The code for generating this table is available at (Finlayson and Kulkarni, 2011b)

Acknowledgments

This work was supported in part by the AFOSR under grant number A9550-05-1-0321, and DARPA under award FA8750-10-1-0076.

References

Victoria Arranz, Jordi Atserias, and Mauro Castillo. 2005. Multiwords and word sense disambiguation. In Alexander Gelbukh, editor, *Proceedings of the Sixth International Conference on Intelligent Text Processing and Computational Linguistics (CICLING 2005)*, volume 3406 in Lecture Notes in Computer Science (LNCS), pages 250–262, Mexico City, Mexico. Springer-Verlag.

Satanjeev Banerjee and Ted Pedersen. 2003. The design, implementation, and use of the ngram statistics package. In Alexander Gelbukh, editor, *Proceedings of the Fourth International Conference on Intelligent Text Processing and Computational Linguistics (CICLING 2003)*, volume 2588 in Lecture Notes in Computer Science (LNCS), pages 370–381, Mexico City, Mexico. Springer-Verlag.
`http://ngram.sourceforge.net`.

Christiane Fellbaum. 1998. *Wordnet: An Electronic Lexical Database*. MIT Press, Cambridge, MA.

Mark Alan Finlayson and Nidhi Kulkarni. 2011a. jMWE:, version 1.0.0.
`http://projects.csail.mit.edu/jmwe`
`http://hdl.handle.net/1721.1/62793`.

Mark Alan Finlayson and Nidhi Kulkarni. 2011b. Source code and data for MWE'2011 papers.
`http://hdl.handle.net/1721.1/62792`.

Mark Alan Finlayson. 2008a. JSemcor, version 1.0.0.
`http://projects.csail.mit.edu/jsemcor`.

Mark Alan Finlayson. 2008b. JWI: The MIT Java Wordnet Interface, version 2.1.5.
`http://projects.csail.mit.edu/jwi`.

Carlos Ramisch, Aline Villavicencio, and Christian Boitet. 2010. Multiword expressions in the wild? the mwetoolkit comes in handy. In Chu-Ren Huang and Daniel Jurafsky, editors, *Proceedings of the Twenty-Third International Conference on Computational Linguistics (COLING 2010): Demonstrations*, volume 23, pages 57–60, Beijing, China.
`http://mwetoolkit.sourceforge.net`.

Kristina Toutanova, Daniel Klein, Christopher D. Manning, and Yoram Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL)*, pages 252–259, Edmonton, Canada.