Shift-reduce based RST Parser Version 0.01

Yangfeng Ji

Sep. 25th, 2014

1 RST based Discourse Processing

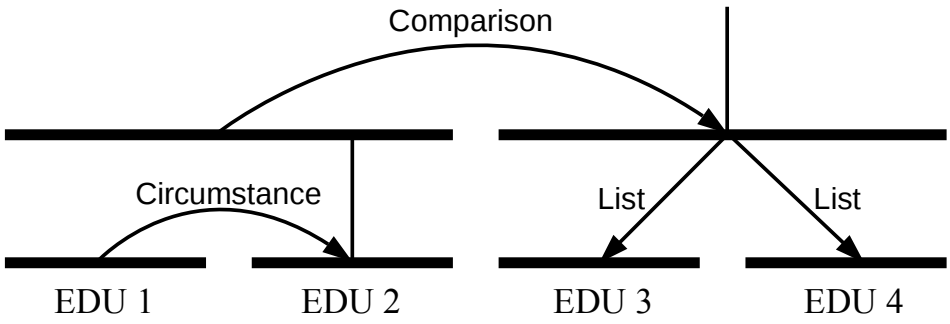

Figure 1: An example of RST tree.

2 Shift-reduce Parsing

Shift-reduce parsing has been used in discourse parsing for more than 10 years [2, 4, 5]. As explained in [2], the way of choosing a specific parsing action can be viewed as a multi-class classification problem. In the classification based parsing framework, each specific parsing action corresponds one classification label. During parsing procedure, an multi-class classifier takes generated feature as input and outputs one specific parsing action.

3 Code

3.1 Demo

Run code ***main.py*** for a demo

3.2 Code Structure

- ***tree.py***: any operation about an RST tree is included in this module. For example
  - Build general/binary RST tree from annotated file
  - Binarize a general RST tree to the binary form (original RST trees in the RST treebank may not in the binary form)
  - Generate bracketing sequence for evaluation
  - Write an RST tree into file (not implemented yet)
  - Generate Shift-reduce parsing action examples
  - Get all EDUs from the RST tree
- ***parser.py***: an implementation of the shift-reduce parsing algorithm, including following functions:
  - Initialize parsing status given a sequence of texts
  - Change the status according to a specific parsing action
  - Get the status of stack/queue
  - Check whether should stop parsing
- ***model.py***: an parsing model module, where a trained parsing model could predict parsing actions. This module includes:
  - Batch training on the data generated by the data module
  - Predict parsing actions for a given feature set
  - Save/load parsing model
- ***feature***: an feature generator, which can generate features from current stack/queue status.
- ***data***: generate training data for offline training

3.3 Data structure

In RST parser, one important data structure for the RST parser is SpanNode, which is defined in ***datastructure.py***. It includes following properties:

- text: the text of this span
- relation: discourse relation between this span and its sibling
- eduspan: edu range in this span
- nucspan: nucleus range in this span (not clear enough)

- prop: property of this span with respect to its parent node. It only has two possible values: Nucleus or Satellite
- lnode/rnode: left/right children node, which are also instances of class SpanNode
- pnode: parent node of this span, which is also an instance of class SpanNode
- nodelist: (only used for general RST tree)
- form: the relation between this span and its sibling

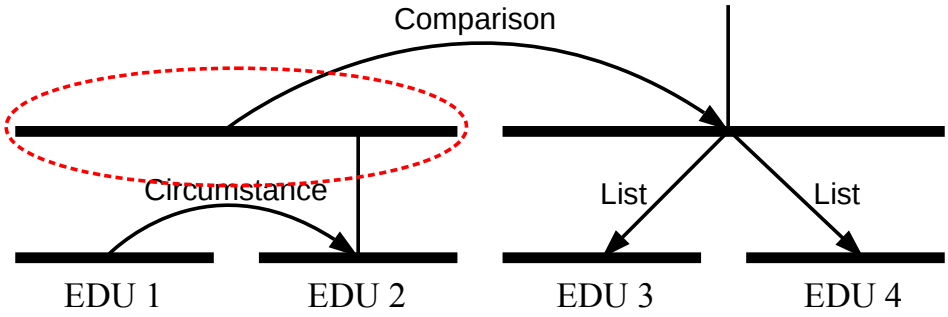

Figure 2: The example tree for explaining the data structure SpanNode.

Taking the circled node in Figure 2 as an example, it is a discourse span consisting of two child node: EDU 1 and EDU 2. For this node, the values of those properties are:

- text: the combination of text from EDU 1 and EDU 2
- relation: Comparison
- eduspan: (1,2)
- nucspan: (2,2)
- prop: Nucleus — considering its sibling, which is a span consisting of EDU 3 and EDU 4, the circled node is a nucleus span while its sibling is a satellite span
- lnode/rnode: EDU 1 / EDU 2
- pnode: the root node
- form: NS — as the left child node of its parent, it is a nucleus span. Therefore, the form is NS (Nucleus-Satellite)

3.4 Feature Generator

The feature generator is defined in ***feature.py***. The main purpose is to provide an unified feature generating process for training data and test data. The advantage of using an unified interface is to guarantee the consistency of the feature set.

Usually, for shift-reduce parsing, features are extract from the top two elements from the stack and the first element from the queue. Because those three elements are related to the decision making on parsing actions.

In class FeatureGenerator, there are four variables used within this class:

- span1: the **top first** element in the **stack**, which is an instance of SpanNode. If the stack is empty, it should be None.
- span2: the **top second** element in the **stack**, which is an instance of SpanNode. If the stack only has one element, it should be None.
- span3: the **first** element in the **queue**. It is an instance of SpanNode and also a single EDU. If the queue is empty, it should be None.
- doclen: document length with respect to number of EDUs (As all document information could be inferred from the status of stack/queue, this variable could be removed in future).

one major function and three sub-functions:

- features(): the major function to generate a set of features according the current status of stack and queue.
- lexical_features(): extracting n-gram features or some other lexical features related to n-gram, for example, POS tags of n-gram.
- structural_features(): structural information about one element (from either the top of stack or the head of queue) with respect to the entire document. For example, the distance from the beginning of document to the current span.
- status_features(): information about the current status of stack/queue. For example, how many element in stack — if there is only one element in stack, then definitely a shift action is required in the next parsing step.

As you may realize, there are tons of features could be useful for discourse parsing [1, 3] and lots of them are missed from the current implementation. You are welcome to add any feature which you think are useful.

3.5 Evaluation

References

[1] Vanessa Wei Feng and Graeme Hirst. Text-level Discourse Parsing with Rich Linguistic Features. In *Proceedings of ACL*, 2012.

[2] Yangfeng Ji and Jacob Eisenstein. Representation Learning for Text-level Discourse Parsing. In *ACL*, 2014.

[3] Shafiq Joty, Giuseppe Carenini, Raymond Ng, and Yashar Mehdad. Combining Intra- and Multi-sentential Rhetorical Parsing for Document-level Discourse Analysis. In *Proceedings of ACL*, 2013.

[4] Daniel Marcu. A Decision-Based Approach to Rhetorical Parsing. In *Proceedings of ACL*, pages 365–372, College Park, Maryland, USA, June 1999. Association for Computational Linguistics.

[5] Kenji Sagae and Alon Lavie. A Classifier-based Parser with Linear Run-Time Complexity. In *Proceedings of the Ninth International Workshop on Parsing Technology (IWPT)*, pages 125–132, 2005.