Procesamiento del Lenguaje Natural, Revista nº 53, septiembre de 2014, pp 181-184 recibido 10-04-14 revisado 07-07-14 aceptado 07-07-14

ViZPar: A GUI for ZPar with Manual Feature Selection

ViZPar: una GUI para ZPar con Selección Manual de Features

Isabel Ortiz[1], Miguel Ballesteros[2] and Yue Zhang[3]
[1]Pompeu Fabra University, Barcelona, Spain
[2]Natural Language Processing Group, Pompeu Fabra University, Barcelona, Spain
[3]Singapore University of Technology and Design
[1]iortiztornay@gmail.com
[2]miguel.ballesteros@upf.edu, [3]yue_zhang@sutd.edu.sg

Resumen: Los analizadores de dependencias y constituyentes se utilizan masivamente en la comunidad de Procesamiento de Lenguaje Natural. ZPar implementa versiones precisas y eficientes de algoritmos shift-reduce para parsing. En este artículo se presenta ViZPar, que es una interfaz gráfica de usuario para ZPar incluyendo visualización de árboles y selección automática de features. Durante la sesión de demostración se ejecutará ViZPar, para dependencias y constituyentes, y explicaremos las funcionalidades del sistema.
Palabras clave: Análisis de dependencias, Análisis de constituyentes, ZPar

Abstract: Phrase-structure and dependency parsers are used massively in the Natural Language Processing community. ZPar implements fast and accurate versions of shift-reduce dependency and phrase-structure parsing algorithms. We present ViZPar, a tool that enhances the usability of ZPar, including parameter selection and output visualization. Moreover, ViZPar allows manual feature selection which makes the tool very useful for people interested in obtaining the best parser through feature engineering, provided that the feature templates included in ZPar are optimized for English and Chinese. During the demo session, we will run ViZPar for the dependency and the phrase-structure versions and we will explain the potentialities of such a system.
Keywords: Dependency parsing, Phrase-Structure parsing, ZPar

1 Introduction

Natural language researchers and application developers apply dependency and constituency parsing frequently, however the parsers normally require careful tuning, complex optimization and the usage of complex commands that hinder their use.

ZPar[1] is a state-of-the-art parser implemented in C++ focused on efficiency. It provides a dependency parser (Zhang and Nivre, 2011; Zhang and Nivre, 2012) and a phrase-structure parser (Zhang and Clark, 2011a; Zhu et al., 2013), both implemented by shift-reduce parsing algorithms (Nivre, 2003; Nivre, 2008; Sagae and Lavie, 2005). ZPar gives competitive accuracies and very fast parsing speeds on both tasks. However, ZPar requires deep knowledge in command-line interfaces and programming skills (especially if the user is interested in performing feature engineering), which leaves its use to researchers that are able to understand the intricacies of such a system. As a result it is relatively more difficult to use by corpus linguists and researchers who need to apply syntactic analysis, but are not familiar with parsing research.

The usability limitation applies to other parsers also, including the Collins parser (Collins, 1999) or MaltParser[2] (Nivre et al., 2007), as it is a common practice for statistical parsers to use command-line interfaces. On the other hand, there has been a call for enhanced usability of parsers,[3] and visualization tools and application wrappers; they have made a non negligible impact on the parsing research field.

In order to have a system that tries to en-

[1]http://sourceforge.net/projects/zpar/

[2]MaltParser has been one of the most widely used parsers, since there are existing parallel tools, including visualization tools.

[3]See Section 4.

ISSN 1135-5948

hance the usability of ZPar, we present ViZPar,[4] which is a tool implemented in Java that provides a graphical user interface of ZPar, in its dependency and phrase-structure versions. ViZPar also allows manual feature engineering given the ZPar feature templates and provides automatic evaluation and comparison tools with the gold-standard.

2 ZPar

ZPar is a statistical syntactic analyzer that performs tokenization/segmentation, POS-tagging, dependency parsing and constituent parsing functionalities. ZPar is language-independent but contains optimized versions for the Penn Treebank (Marcus, Santorini, and Marcinkiewicz, 1993) and the Chinese Treebank (Xue et al., 2004). Currently, in its out of the box version, it gives highly competitive accuracies on both English (Zhu et al., 2013) and Chinese (Zhang et al., 2013) benchmarks.

ZPar is implemented using the shift-reduce parsing mechanism (Yamada and Matsumoto, 2003; Nivre, 2008; Sagae and Lavie, 2005). It leverages a global discriminative training and beam-search framework (Zhang and Clark, 2011b; Zhang and Nivre, 2012) to improve parsing accuracies while maintaining linear time search efficiency. As a result ZPar processes over 50 sentences per second for both constituent parsing and dependency parsing on standard hardware. It is implemented in C++, and runs on Linux and MacOS. It provides command-line interfaces only, which makes it relatively less useful for researchers on corpus linguistics than for statistical parsing researchers.

3 ViZPar: a Vizualization tool for ZPar

ViZPar is a graphical user interface of ZPar implemented in Java. In its current version it supports a GUI for training and using ZPar, including the visualization of dependency and constituent outputs, evaluation and comparison with gold-standard treebanks, manual configuration and feature selection.

3.1 Java Wrapping of ZPar

The ZPar package includes a bash script that compiles the C++ code, trains a parsing model, runs the parser over the development set and finds the results of the best iteration. ViZPar provides a graphical-user interface for running the ZPar process, which wraps the entire process with a graphical user interface.

A user of ViZPar needs to download ZPar, which contains the C++ source code and the Python and bash scripts needed. On initialization, ViZPar asks for the ZPar directory, which is needed to train models and run the parser. After that, the user should proceed as follows:

- Selection of parsing mode: the user selects whether he/she wants to run a dependency parser or a constituent parser.
- Selection of mode: the user may select an existing parsing model or train a new one by also setting the number of iterations.
- Selection of the test set for parsing and evaluation.

Finally, when the process is finished the system allows to visualize the output by using graphical visualization of the syntactic trees. This feature is explained in the following section.

3.2 Tree Visualization

In order to visualize the output and the gold standard trees of the dependency and phrase structure versions we implemented two different solutions. For the dependency parsing version, we reused the source code of MaltEval (Nilsson and Nivre, 2008) for tree visualization, which includes all its functionalities, such as zooming or digging into node information, and for the constituent parsing version, we implemented a completely new tree visualizer.

Figure 1 shows the ViZPar graphical user interface when it has already parsed some dependency trees with a model trained over an English treebank. The dependency tree shown at the top of the picture is the gold standard and the one shown below is the output provided by the ZPar model. In the same way, Figure 2 shows ViZPar GUI in the case of the phrase-structure parser, which allows to traverse the tree and check the outcome quality by comparison with the gold-standard.

[4] ViZPar stands for graphical visualization tool for ZPar.

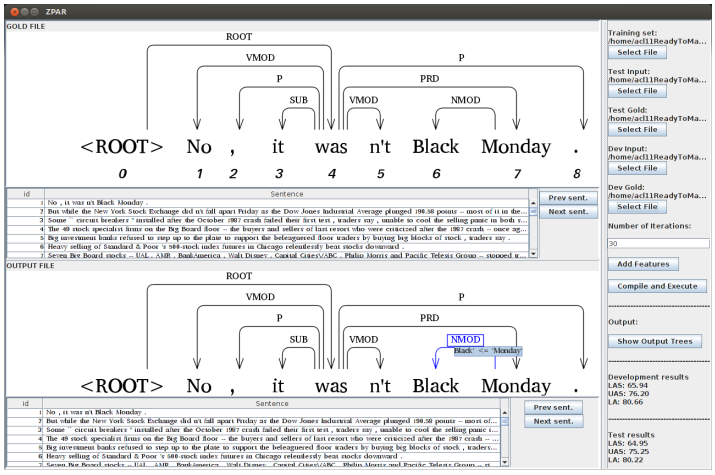

Figure 1: ViZPar in 'dependency' mode.

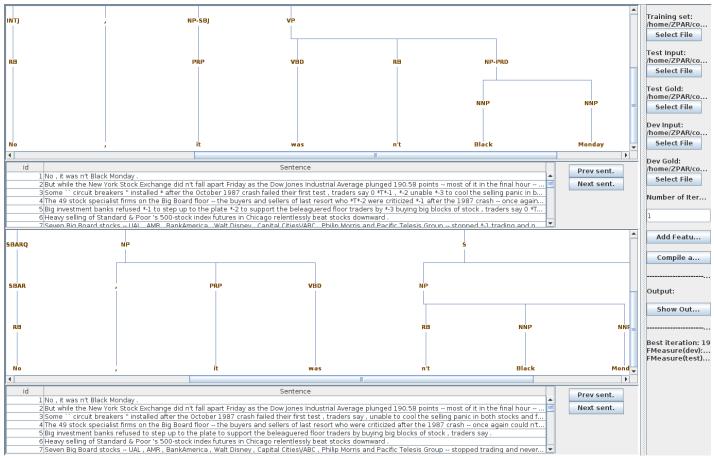

Figure 2: ViZPar in 'constituent' mode.

3.3 Feature Selection

ZPar provides rich feature templates, depicted by Zhang and Nivre (2011; 2012) for dependency parsing and by Zhu et al. (2013) for constituent parsing. However the features are handcrafted in the source code; this fact means that if the users would like to update the set of features for a new language or treebank, they would have to change the source code, compile it and run it again, provided that the user knows where the feature templates are and how to encode them. In ViZPar we provide a framework that allows the selection of the different features, and changes the source code automatically, it also makes the compilation and outputs the parser ready to generate a new model.

Our algorithm is implemented by scanning through the ZPar source code, detecting lines on feature definition, which follow regular patterns, and listing the features in a dialog box. The algorithm changes the ZPar source code according to the user selection by commenting out features that are deselected.

The manual feature selection tool might also provide the opportunity of running automatic and manual feature selection experiments as in MaltOptimizer (Ballesteros and Nivre, 2012).

4 Related Work

There has been recent research on visualization in the NLP community. In the parsing area we can find systems, such as MaltEval (Nilsson and Nivre, 2008), which allows the comparison of the output with a gold standard and also includes statistical significance tests. The Mate Tools (Bohnet, Langjahr, and Wanner, 2000) provide a framework for generating rule-based transduction and visualization of dependency structures. Icarus (Gartner et al., 2013) is a search tool and visualizer of dependency treebanks. Finally,

MaltDiver (Ballesteros and Carlini, 2013) visualizes the transitions performed by MaltParser.

5 Conclusions

In this paper we have presented ViZPar which is a Java graphical user interface of ZPar. We have shown its main functionalities, that are: (1) run ZPar in a user friendly environment, (2) dependency and constituent tree visualization and (3) manual feature engineering. ViZPar can be downloaded from `http://taln.upf.edu/system/files/resources_files/ViZPar.zip`

References

Ballesteros, Miguel and Roberto Carlini. 2013. MaltDiver: A Transition-Based Parser Visualizer. In *Proceedings of the System Demonstration Session of the 6th International Joint Conference on Natural Language Processing (IJCNLP)*.

Ballesteros, Miguel and Joakim Nivre. 2012. MaltOptimizer: A System for MaltParser Optimization. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC)*.

Bohnet, Bernd, Andreas Langjahr, and Leo Wanner. 2000. A development environment for an mtt-based sentence generator. In *Proceedings of the First International Natural Language Generation Conference*.

Collins, Michael. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania.

Gartner, Markus, Gregor Thiele, Wolfgang Seeker, Anders Bjorkelund, and Jonas Kuhn. 2013. Icarus – an extensible graphical search tool for dependency treebanks. In *ACL-Demos*, August.

Marcus, Mitchell P., Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19:313–330.

Nilsson, Jens and Joakim Nivre. 2008. Malteval: an evaluation and visualization tool for dependency parsing. In *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco, may.

Nivre, J. 2003. An Efficient Algorithm for Projective Dependency Parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pages 149–160.

Nivre, J., J. Hall, J. Nilsson, A. Chanev, G. Eryiğit, S. Kübler, S. Marinov, and E. Marsi. 2007. Maltparser: A Language-Independent System for Data-Driven Dependency Parsing. *Natural Language Engineering*, 13:95–135.

Nivre, Joakim. 2008. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34:513–553.

Sagae, Kenji and Alon Lavie. 2005. A classifier-based parser with linear runtime complexity. In *Proceedings of the 9th International Workshop on Parsing Technologies (IWPT)*, pages 125–132.

Xue, Naiwen, Fei Xia, Fu-Dong Chiou, and Martha Palmer. 2004. The Penn Chinese Treebank: Phase structure annotation of a large corpus. *Journal of Natural Language Engineering*, 11:207–238.

Yamada, Hiroyasu and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pages 195–206.

Zhang, Meishan, Yue Zhang, Wanxiang Che, and Ting Liu. 2013. Chinese parsing exploiting characters. In *ACL*.

Zhang, Yue and Stephen Clark. 2011a. Shift-reduce ccg parsing. In *ACL*.

Zhang, Yue and Stephen Clark. 2011b. Syntactic processing using the generalized perceptron and beam search. *Computational Linguistics*, 37(1):105–151.

Zhang, Yue and Joakim Nivre. 2011. Transition-based dependency parsing with rich non-local features. In *ACL (Short Papers)*, pages 188–193.

Zhang, Yue and Joakim Nivre. 2012. Analyzing the effect of global learning and beam-search on transition-based dependency parsing. In *COLING (Posters)*.

Zhu, Muhua, Yue Zhang, Wenliang Chen, Min Zhang, and Jingbo Zhu. 2013. Fast and accurate shift-reduce constituent parsing. In *ACL*.