Using Clojure in Linguistic Computing

Zoltán Varjú
Weblib LLC
Kaposhegy u. 13
7400 Kaposvar, Hungary
zoltan.varju@weblib.com

Richard Littauer
Saarland University
Computational Linguistics
Department
66041 Saarbrücken, Germany
richard.littauer@gmail.com

Peteris Erins
University of Cambridge
Selwyn College
Cambridge, United Kingdom
peteris.erins@gmail.com

ABSTRACT

Due to recent developments in the humanities and the social sciences more and more scholars are turning towards computing. Although there are accessible and well designed toolkits to get into computing, taking the step from surface knowledge of programming to tackling non-trivial issues is difficult and time intensive. Even those researchers with some computing background may face problems when they want to deepen their knowledge and start serious projects, especially when participating with a team of programmers is not an option. We think using Clojure for linguistic computing answers these concerns. Here, we examine the existing resources and explain why Clojure can help in both educating and researching. We also present two mini case studies as examples of Clojure's interoperability and use.

Categories and Subject Descriptors

A.m [**General Literature**]: Miscellaneous; I.7.m [**Computing Methodologies**]: Document and Text Processing—*Miscellaneous*

General Terms

Theory

Keywords

Clojure, linguistic computing

1. INTRODUCTION

Computational methods are standard tools in the natural sciences, and there is an increasing demand for easy to use tools in the emerging fields of digital humanities, linguistic computing, computational social sciences and data journalism. In these fields, the practice of scientific investigation relies on computational methods. Analyzing data and running simulations are inherent parts of current scientific work that require programming skills.

Although there is a plethora of high quality open source projects, the average researcher must become a polyglot and use various programming languages to achieve their goals. This results, far too often, in a long and steep learning curve, slowly progressing research projects, and bad programming practices. There is no royal road to computer science. Those who want to step into emerging new fields do not have to become computer scientists, but they have to acquire the basics in order to participate in interdisciplinary projects and conduct their own research.

The Clojure language is ideal for the above mentioned scenario for several reasons. The theoretical basis of functional programming can be related to the humanities curriculum. The clear and accessible Clojure development environment encourages good software engineering practice. Clojure, as a JVM language, and with its unique mechanism for parallelism, is a natural choice in the age of the data deluge.

2. EXISTING RESOURCES

The most popular natural language processing (NLP) tool is the Python Natural Language Toolkit (`http://www.nltk.org/`) (NLTK). It contains a wide range of algorithms, both statistical and rule based. It was designed for multidisciplinary instruction[3] and it is used at institutions all over the world. Today, NLTK is not only an academic toolkit; it is commonly used in the industry too. The freely available NLTK book[4] and popular titles from big publishers[18] paired with its wide range of algorithms make NLTK an outstanding NLP tool.

The Stanford CoreNLP(`http://nlp.stanford.edu/software/corenlp.shtml`) and the Apache OpenNLP(`http://incubator.apache.org/opennlp/index.html`) projects are the two most reliable and advanced Java-based NLP tools. Although their quality is unquestionable, using them require advanced knowledge of the Java language and object oriented programming methodology. The lack of good tutorials and comprehensive documentation mean a barrier to non-programmers.

SNLTK(`http://www.snltk.org/`) is the most comprehensive Lisp-based NLP library. It is designed for educational purposes, building on the rich ecosystem around Racket and Scheme. SNLTK inspired us as its authors emphasize the rich Lisp literature on the foundations computational Linguistics[8].

Prolog once was a prominent language among computational linguists. Logic programming is ideal for expressing foun-

dational concepts like automata theory and one can easily implement basic grammar formalisms in Prolog. A rich literature[6, 5] developed around the language over the years but the decline of symbolic AI made Prolog less popular, leaving it unaccessible for a wider audience.

In the last decade, Haskell has emerged as a new tool for those who are interested in applying computational methods to logic and semantics. Titles like [10] and [20] laid down foundations. Natural Language Processing for the Working Programmer[9] is a work-in progress open book and a proof of concept of the usage of Haskell for statistical NLP.

The R statistical programming language is very popular among corpus linguists, psycholinguists and social scientists. Packages like tm[12] and zipfR[11] are not only valuable educational and research tools, but can be used in applied research too. Despite its merits, it is criticized because of its steep learning curve and inherent problems with parallelism and scalability[15].

3. MOTIVATIONS FOR CHOOSING CLOJURE

We believe that Clojure is a good choice as a second programming language for scholars and students in the humanities and social sciences. At present, we would like to test our assumptions through mini case studies.

Norvig in his seminal Paradigms of Artificial Intelligence Programming[17] lists eight features that make Lisp different and the natural choice for artificial intelligence programming: built-in support for lists, automatic storage management, dynamic typing, first-class functions, uniform syntax, interactive environment, extensibility, and history.

Clojure not only supports list, but other common data structures. Every popular IDE has Clojure support and emacs with Slime(`http://common-lisp.net/project/slime/`) or VimClojure(`http://www.vim.org/scripts/script.php?script_id=2501`) provide users with a high quality development environment. Java interoperability means more extensibility with the abundance of Java libraries. We have to add to the list the open source community built around the language, which made tools like the ClojureScript(`https://github.com/clojure/clojurescript`) and the ClojureScriptOne(`http://clojurescriptone.com/`) web development environment and the pallet tool for cloud computing(`http://palletops.com/`) which reduces the learning curve as one get things done in a language.

As an extensible language, Clojure is multiparadigmatic. Type checking is optional, and core.logic implements miniKanren, a Prolog-like logic programming library.

3.1 Pedagogical considerations

Although the curriculum is changing, some elementary logic is usually part of the humanities education, at least in philosophy and linguistics programs. Social scientists have to acquire statistics and get used to a basic level of mathematical rigour during their studies. Given this background and some exposure to the fundamental concepts of programming they are in a very good position to learn 21st century computing skills that can be useful in academic research and even in the job market.

However the above mentioned type of people have to face serious problems when they want to deepen their knowledge. Although we have well-written accessible books on computational linguistics they are either outdated or one should learn at least two programming languages in parallel to get things done.

3.2 Scientific considerations

Computing is going through a big change[13]. The available data grows exponentially; "today's big data is tomorrow's mid-size" as the saying goes[19]. New technologies, and even new professions like data scientist have emerged in the last few years.

As the Google n-gram viewer(`http://books.google.com/ngrams`) project shows we can have access to unimaginable amount of data[16]. To work effectively and be able to collaborate with colleagues from other fields, humanities researchers need a tool that is smoothly scalable.

While Linguistics is not traditionally a data-intensive science, with the increasing size of linguistic databases, it is nevertheless entering the era of the fourth paradigm[14], where managing and accessing data is as integral a part of the scientific process as collecting it using computational methods. These unprecedented possibilities urge linguists to build a cyber-infrastructure[2] and rethink their theories[1] in the light of data.

Using a tool, like Clojure, that addresses these issues could foster scholarly work of a new type, and especially interdisciplinary work.

4. MINI CASE STUDIES

We started a project blog where we are showing off possible uses of Clojure and we are trying to get feedback from Clojure enthusiast and members of the humanities computing community. We found two types of interested individuals; the first is using computational methods as a tool during their analysis, the second is interested in using the computational tools to express their ideas.

4.1 Java interoperability

The first group sees programming as a tool that helps analysis. This does not mean that they compromise quality for usability. This group can benefit from the interoperability of Clojure with high quality Java libraries like OpenNLP. A corpus linguist interested in the distribution of various parts of speech elements can use a POS tagger to automatically tag collections of raw electronic texts. As a common software engineering practice, one can use a tool as a kind of black box without any expert knowledge about its internal mechanism.

OpenNLP can be integrated into a Clojure project via Leiningen. One can easily define the necessary tools to POS tag sentences with a few line of code.

```
(ns hello-nlp.core
    (use opennlp.nlp)
    (use opennlp.treebank)
    (use opennlp.tools.filters)
    (use (incanter core charts)))

(def get-sentences
    (make-sentence-detector "models/en-sent.bin"))

(def tokenize
    (make-tokenizer "models/en-token.bin"))

(def pos-tag
    (make-pos-tagger "models/en-pos-maxent.bin"))
```

There is no need for expert knowledge to get started with counting parts of speech. Thinking about linguistic data structures in terms of lists (or sets and hash-maps) is a natural choice, as using functions on these structures is close to linguistic formalism. Counting parts of speech in a text can be expressed in an easy to read manner.

```
(defn tag-sent [sent]
    (pos-tag (tokenize sent)))
(def pos-austen
    (map pos-tag (map tokenize (get-sentences austen))))

(pos-filter determiners #"^DT")
(pos-filter prepositions #"^IN")

(def preps
    (reduce + (map count (map prepositions pos-austen))))
(def dets
    (reduce + (map count (map determiners pos-austen))))
(def nps
    (reduce + (map count (map nouns pos-austen))))
(def vps
    (reduce + (map count (map verbs pos-austen))))
(def stats
    [nps vps dets preps])

(view (bar-chart ["np" "vp" "dts" "preps" ] stats))
```

The output graph can be seen in Fig. 1.

Collecting and presenting word frequency distributions is a tedious task. Linguists often use pipelines of various tools, one for collecting and cleaning up texts and another for analyzing and report findings. This works requires using wrappers of third party tools like part of speech taggers. The aforementioned process is called ”software carpentry”, which describes the pragmatic approach of those who see software as a means, not an end. But as we have described in Section 3.2, the scale of scientific data is changing and traditional tools for processing it are becoming insufficient. While the product of ”carpentry” can be used as prototypes, we can gain in productivity by using interoperable tools.

In the future, the ability to work together with computer scientists will be more important. Product cycles are getting shorter and shorter, the border of prototyping and production is disappearing and parallelism is becoming common.

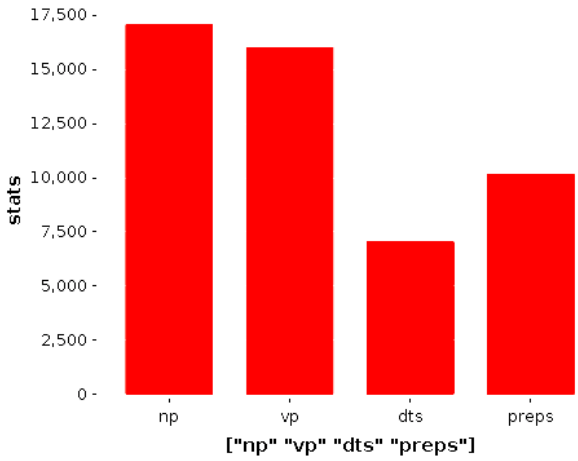

Figure 1: Word tag frequencies

With Clojure, one can use Java or Clojure libraries developed by the project team or foundnd in a public open source repository. Upon completing the work, the end result should ready for integration.

4.2 Logic programming

Logic programming is a programming paradigm that uses mathematical logic for writing programs. Logic programs are classically written as sets of inference rules and a query over the rule database. The execution of a logic program is an attempt to constructively prove the query using a built-in proof-search method.

Many concepts in computational linguistics, such as automata and grammars, can be described declaratively. Logic programming languages can be used to turn their definitions into computations. Linguists can write recognizers, parsers and generators without implementing a search algorithm.

Prolog has been the dominant logic programming language in linguistics and beyond. Its use has diminished over the years; however, the premise of logic programming still stands. The recent language Kanren is an embedded logic programming environment in Scheme. A version of the library called miniKanren[7] has been ported to Clojure in the form of the core.logic library.

Below we implement an automata in core.logic that accepts lists consisting of multiple copies of the letter a, as seen in Fig. 2.

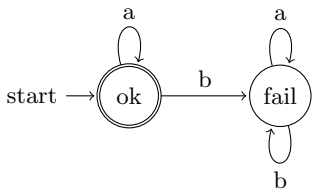

Figure 2: Example automata

```
(defrel start q)
(fact start 'ok)


(defrel transition from via to)
(facts transition [['ok 'a 'ok]
                   ['ok 'b 'fail]
                   ['fail 'a 'fail]
                   ['fail 'b 'fail]])

(defrel accepting q)
(fact accepting 'ok)

(defn recognize
  ([input]
     (fresh [q0]
            (start q0)
            (recognize q0 input)))
  ([q input]
     (matche [input]
        (['()]
           (accepting q))
        ([[i . nput]]
           (fresh [qto]
                  (transition q i qto)
                  (recognize qto nput))))))
```

We can use the automata to recognize strings in the language, or even to generate them.

```
(run* [q] (recognize '(a a a)))
;; => (_.0) where "_0" implies success (run* [q] (recognize '(a b a)))
;; => ()

(run 3 [q] (recognize q))
;; => (() (a) (a a))
```

Embedding in a functional language permits the use of logic programming just when appropriate. That is an advantage over Prolog, which does not feature functional constructs.

5. FUTURE WORK

Our primary goal is to provide more mini-case studies and get more feedback from a wider audience. Ultimately, we would like to combine the case studies into a toolkit that can be used for linguistics education and research.

6. REFERENCES

[1] Bender, E. M., and Good, J. A grand challenge for linguistics: Scaling up and integrating models. *White paper contributed to NSF's SBE 2020: Future Research in the Social, Behavioral and Economic Sciences initiative 1*, 1 (June 2010), 1–1.

[2] Bender, E. M., and Langendoen, D. T. Computational linguistics in support of linguistic theory. *Linguistic Issues in Language Technology 3*, 2 (February 2010), 0–0.

[3] Bird, S., Klein, E., Loper, E., , and Baldridge, J. Multidisciplinary instruction with the natural language toolkit. In *Proceedings of the Third Workshop on Issues in Teaching Computational Linguistics* (June 2008), Third Workshop on Issues in Teaching Computational Linguistics, pp. 62–70.

[4] Bird, S., Klein, E., and Loper, E. *Natural Language Processing with Python*. O'Reilly Media, Sebastopol, CA, 2009.

[5] Blackburn, P., and Bos, J. *Representation and Inference for Natural Language: A First Course in Computational Semantics*. Center for the Study of Language and Information, Palo Alto, CA, 2005.

[6] Blackburn, P., Bos, J., and Striegnitz, K. *Learn Prolog Now*. College Publications, London, UK, 2006.

[7] Byrd, W. E. *Relational Programming in miniKanren: Techniques, Applications, and Implementations*. PhD thesis, Indiana University, August 2009.

[8] Cavar, D., Gulan, T., Kero, D., Pehar, F., and Valerjev, P. The scheme natural language toolkit (snltk). In *Proceedings of the 4th European Lisp Symposium* (April 2011), European Lisp Symposium, pp. 58–61.

[9] de Kok, D., and Brouwer, H. *Natural Language Processing for the Working Programmer*. `http://nlpwp.org/`, Groningen, NL, 2011.

[10] Doets, K., and van Eijck, J. *The Haskell Road to Logic, Maths and Programming*. College Publications, London, UK, 2004.

[11] Evert, S., and Broni, M. zipfr: Word frequency distributions in R. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics* (2007).

[12] Feinerer, I., Hornik, K., and Meyer, D. Text mining infrastructure in R. *Journal of Statistical Software 25*, 1 (March 2008), 1–54.

[13] Halvey, A., Norvig, P., and Pereira, F. The unreasonable effectiveness of data. *IEEE Intelligent Systems 24*, 2 (March/April 2009), 8–12.

[14] Hey, T., Tansley, S., and Tolle, K., Eds. *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Microsoft Corporation, US, 2009.

[15] Ihaka, R., and Lang, D. T. Back to the future: Lisp as a base for a statistical computing system. *Compstat 2008 24*, 1-1 (August 2008), 1–1.

[16] Michel, J.-B., Shen, Y. K., Aiden, A. P., Veres, A., Gray, M. K., Pickett, J. P., Clancy, D. H. D., Norvig, P., Orwant, J., Pinker, S., Nowak, M. A., Aiden, E. L., and Team, G. B. Quantitative analysis of culture using millions of digitized books. *Science 331*, 176 (January 2011), 176–182.

[17] Norvig, P. *Paradigms of Artificial Intelligence Programming: Case Studies in Common Lisp*. Morgan Kaufmann, Waltham, MA, 1991.

[18] Russell, M. A. *Mining the Social Web: Analyzing Data from Facebook, Twitter, LinkedIn, and Other Social Media Sites*. O'Reilly Media, Sebastopol, CA, 2011.

[19] Slocum, M. *Big Data Now*. O'Reilly Media, Sebastopol, CA, 2011.

[20] van Eijck, J., and Unger, C. *Computational Semantics with Functional Programming*. Cambridge University Press, Cambridge, UK, 2010.