CL-NLP – A Natural Language Processing Library for Common Lisp

Vsevolod Dyomkin

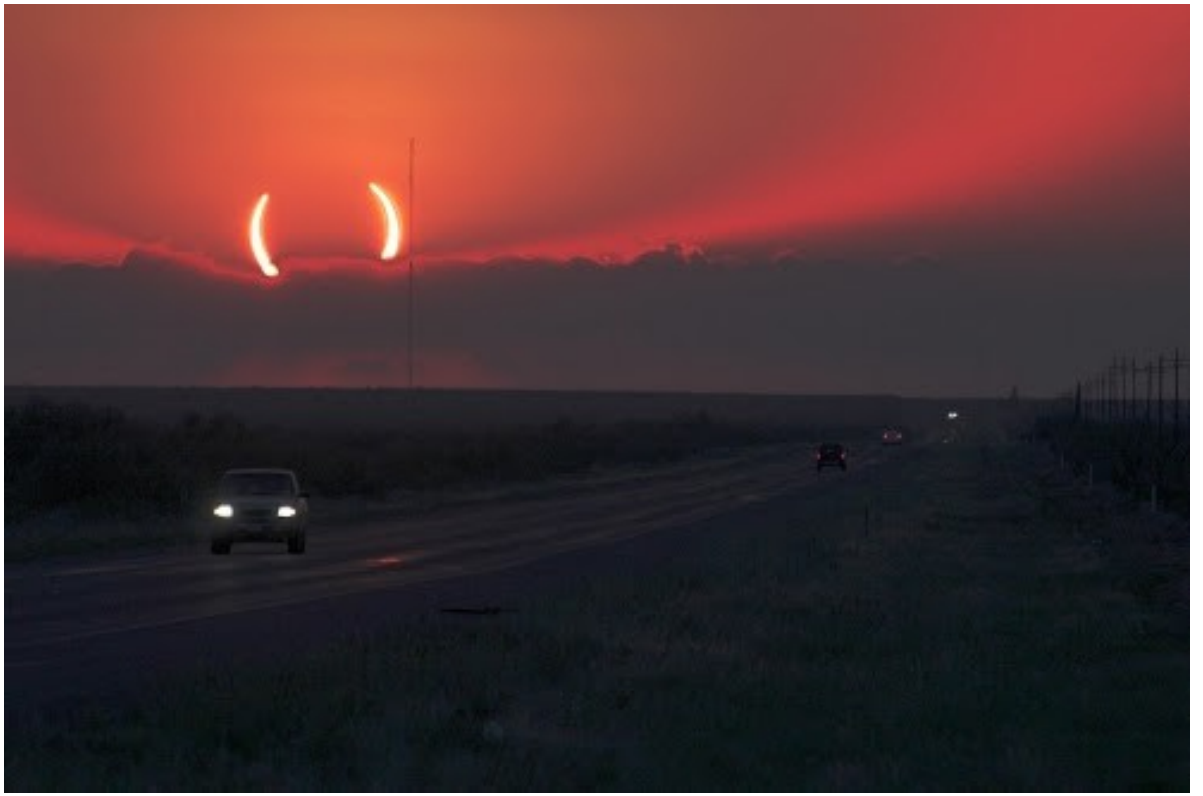
European Lisp Symposium
2013-06-04

Topics

* Motivation
* Library high-level design
* Overview of the functions
* Some technical details
* Next steps

Motivation

NLP & Lisp is a match, made in heaven



Other languages

* for Python: NLTK
* for Java: Stanford CoreNLP and OpenNLP

Previous Work

* CL-Langutils
* Lexiparse, Sparser, CL-EARLY-PARSER, Basic-English-Grammar
* Wordnet interfaces

* CMU AI Repository

* Natural Language Understanding
* Natural Language Processing in Lisp

Essential Scope

* Corpora interfaces
* Language modeling
* Measures
* Tokenization
* Tagging
* Parsing
* Wordnet interface

* Construct a pipeline

Roadmap

Version 0.1: the base system with a test suite and manual

Version 1.0: several well-optimized versions of most algorithms with full documentation

Beyond v.1.0

Another system - meta-nlp

Development methodology

...-driven development

High-level Design

* cl-nlp & cl-nlp-contrib
  & more
* base & functional packages
* nlp-user

Some Design Choices

* A pseudo-hierarchical package design

Modules

* nlp.core, nlp.corpora,
  nlp.util, nlp.test-util

* nlp.syntax, nlp.generation,
  nlp.phonetics

* nlp.contrib.wordnet,
  nlp.contrib.ms-ngrams

Some Design Choices

* Use the whole language
- defclass & defstruct
- hash-tables & alists
- do & loop

Some Design Choices

* Grow the language
  - rutils
    + reader macros
    + shorthands
    + dotable, dolines
  - special-purpose utils

A Special-Purpose Util

```
(define-lazy-singleton word-tokenizer
    (make 'postprocessing-regex-word-tokenizer)
  "Default word tokenizer.")

(defun tokenize-ngram (ngrams str)
  "Transform string STR to a list if necessary
   (depending of order of NGRAMS)."
  (if (> (ngrams-order ngrams) 1)
      (tokenize <word-tokenizer> str)
      str))
```

A Special-Purpose Util

```
(defmacro define-lazy-singleton
    (name init &optional docstring)
  (with-gensyms (singleton)
    `(let (,singleton)
       (defun ,name ()
         ,docstring
         (or ,singleton
             (setf ,singleton ,init)))
       (define-symbol-macro
           ,(mksym name :format "<~A>")
         (,name)))))
```

Some Design Choices

* Use CLOS as a foundation

Basic Cell

```
(defclass regex-word-tokenizer (tokenizer)
  ((regex :accessor tokenizer-regex
          :initarg :regex
          :initform
          (re:create-scanner
           "\\w+|[!\"#$%&'*+,./:;<=>?@^`~…\\(\\)
⟨⟩{}\\[\\|\\]———«»“”‘’¶-]")
          :documentation
          "A simpler variant would be [^\\s]+ —
           it doesn't split punctuation, yet
           sometimes it's desirable."))
  (:documentation
   "Regex-based word tokenizer."))
```

Basic Cell

```
(defmethod tokenize
    ((tokenizer regex-word-tokenizer) string)
  (loop
     :for (beg end)
     :on (re:all-matches (tokenizer-regex
                          tokenizer)
                         string)
     :by #'cddr
     :collect (sub string beg end) :into words
     :collect (cons beg end) :into spans
     :finally (return (values words
                              spans)))
```

Another Example

```
(defgeneric parse (model sentence)
  (:documentation
   "Parse SENTENCE with MODEL.")
  (:method :around (model (sentence string))
   (call-next-method
    model (tokenize <word-tokenizer> string))))

(defgeneric parse-n (model sentence n)
  (:documentation
   "Return N best parse trees of the SENTENCE
    with MODEL.")
  (:method :around (model (sentence string) n)
   (call-next-method
    model (tokenize <word-tokenizer> string) n)))
```

Available functions

* char, words and trees helpers
* measures: entropy, LLR
* language modeling (ngrams)
* word tokenization
  and sentence splitting
* Markov chain-based
  generation

Available functions

* interface to corpora:
  - Brown
  - Reuters
  - Penn Treebank
  - NPS Chat
* tagging: HMM and GLM
* parsing: PCFG & k-best PCFG
* Wordnet interface

Parsing

```
(defmethod parse ((grammar pcfg) (sentence list))
  (CKY (let* ((cur (cons rule (1- s)))
              (l (@ pi0 (1- i) (1- s)
                    (second rule)))
              (r (@ pi0 s (1- j) (third rule)))
              (score (if (and l r)
                         (+ (log q) l r)
                         min)))
         (when (> score (or max min))
           (setf max score
                 arg cur)))))
```

Parsing

```
(defmethod parse :around ((grammar pcfg)
                          (sentence list))
  (with-raw-results
    (values (idx->nts (decode-parse-tree
                       sentence bps 0 last iroot))
            (exp (or (@ pi0 0 last iroot) min))
            pi0
            bps)))
```

```
(macrolet
    ((CKY (&body body)
       `(with-slots (rules nts->idx) grammar
          (let* ((pi0 #{}) (bps #{})  ;; also need to init them
                 (min most-negative-single-float))
            (do ((pos 1 (1+ pos)))
                ((>= pos *sentence-length*))
              (do ((i 1 (1+ i)))
                  ((> i (- *sentence-length* pos)))
                (let ((j (+ i pos)))
                  (dotable (_ k nts->idx)
                    (let (max arg)
                      (do ((s i (1+ s)))
                          ((>= s j))
                        (dotable (rule q rules)
                          (when (and (tryadic rule)
                                     (= k (first rule)))
                            ,@body)))
                      (when (if (listp max) max (> max min))
                        (setf (@ pi0 (1- i) (1- j) k) max
                              (@ bps (1- i) (1- j) k) arg))))))))
            (values pi0 bps)))))
```

```
(declaim (inline @))
(defun @ (m i j k)
  (get# (+ (* i *sentence-length* *nt-count*)
           (* j *nt-count*)
           k)
        m))

(defsetf @ (m i j k) (v)
  `(set# (+ (* ,i *sentence-length* *nt-count*)
            (* ,j *nt-count*)
            ,k)
         ,m ,v))
```

CL-NLP Challenges

* Building a test suite
  - regression tests
  - quality tests
* Model serialization and distribution format
* Implementing all the algorithms (obviously)

Summary

* CL-NLP is a suite of NLP tools allowing to build complex pipelines
* At the current state it is 70% feature-complete. Should be ready by end of '13
* The biggest short-term challenge: robust testing