NLTK

Natural Language Toolkit

Steven Bird & Edward Loper
University of Pennsylvania

The Challenge

- How do you set up a computational linguistics course having a strong practical component?
  - many students are learning to program for the first time
    - Prolog, Perl
  - students must learn how to do low–level "housekeeping" tasks before doing anything interesting
  - not enough time is left to teach the subject itself
  - sometimes just omit a practical component

Requirements on a Programming Language

- shallow learning curve
  - new programmers must get immediate rewards
- support for rapid prototyping
  - we want to avoid the compilation step
- self–documenting code
  - programs must be immediately comprehensible
- support for good programming style
  - it must be easy to write well–structured programs
- graphical user interface
  - the language must have a good, easy–to–use GUI

Python: object–oriented scripting

- shallow learning curve
  - Python was designed to be easily learnt by children
- support for rapid prototyping
  - Python is interpreted, with no compilation step
- self–documenting code
  - Python has been called "executable pseudocode"
- support for good programming style
  - Python is object–oriented (but not punitively so)
- graphical user interface
  - Python has an interface to the *tk* GUI toolkit

NLTK: Python−based CL Courseware

- NLTK: Natural Language Toolkit
  - suite of program modules, reference documents, tutorials and problem sets
  - comprehensive set of base types (tokens, trees, ...) and interfaces
- Development:
  - Sponsored by Mitch Marcus and CIS UPenn
  - Created during a graduate CL course (Fall 2001)
  - 25 students from CS, ling, business school, industry
- Coverage:
  - symbolic & statistical natural language processing
  - annotated corpora and corpus linguistics

NLTK Uses

A. course assignments and projects:

- tweak an existing module to change its behavior
- write a new module supporting an existing interface
- build a complete system out of existing and new modules

B. class demonstrations:

- tedious algorithms (e.g. chart parsing) come to life with online demonstrations
- "what if" and "let's try" − live exploration of the topic

Example: Parsing

- What is it like to teach a course using NLTK?
- Demonstration:
  - two kinds of parsing
  - two ways to use NLTK

A. course projects: chunk parsing

B. class demonstrations: chart parsing

Chunk Parsing: What is it?

a type of *light* or *partial* parsing

- only identify the main constituents of a phrase
- don't build full trees
- applications (IR, IE, TTS)

chunk parsing examples:

- *NP chunking:*
  [I] saw [a tall man] in [the park]
- *VP chunking:*
  The man who [was in the park] [saw me]
- *Prosodic phrase chunking:*
  [I saw] [a tall man] [in the park]

NLTK Chunk Module

Components:

- interface to NLTK tokenizer modules
- a variety of rule types with a high-level interface
- scoring

Example rules:

- ChunkRule('<NN.*>')
- ChinkRule('<VB.>*')
- SplitRule('<NN>', '<DT>')
- MergeRule('<JJ>', '<JJ>')

Steps in the student program

1. load some tagged, chunked text
   - using an existing NLTK module
2. make an unchunked version of this text
   - using an existing NLTK module
3. create a cascade of chunk rules
   - single calls to functions in the chunk module
4. apply the rules to the unchunked text
5. score the result
   1. use an existing NLTK module to compare the two chunkings, and report a numerical score

Example 1: Several rule types

```
cascade = [
  ChunkRule('<DT><NN.*><VB.><NN.*>'),
  ChunkRule('<DT><VB.><NN.*>'),
  ChunkRule('<.*>'),
  UnChunkRule('<IN|VB.*|CC|MD|RB.*>'),
  UnChunkRule("<,|\.|``|''>"),
  MergeRule('<NN.*|DT|JJ.*|CD>', '<NN.*|DT|JJ.*|CD>'),
  SplitRule('<NN.*>', '<DT|JJ>')
]
chunkparser = REChunkParser(cascade)
```

Example 2: Blind statistics

```
cascade = [
  ChunkRule('<\$|CD|DT|EX|PDT
             |PRP.*|WP.*|\#|FW
             |JJ.*|NN.*|POS|RBS|WDT>*')
]
chunkparser = REChunkParser(cascade)
```

Example 3: Chinking

```
cascade = [
  ChunkRule('<.*>+')
  ChinkRule('<VB.*|IN|CC|R.*|MD|WRB|TO|.|,>+')
]
chunkparser = REChunkParser(cascade)
```

Competition Scoring

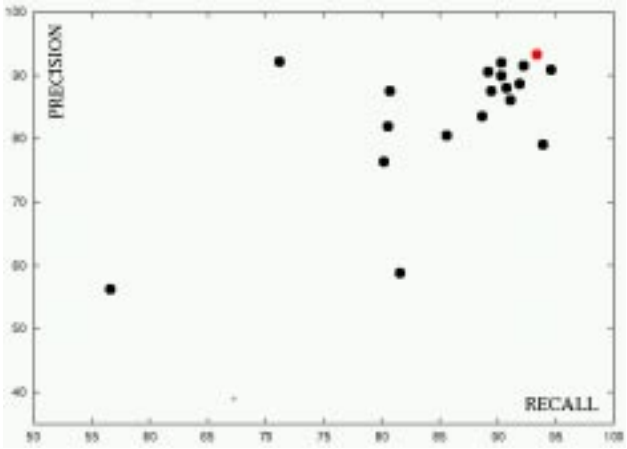
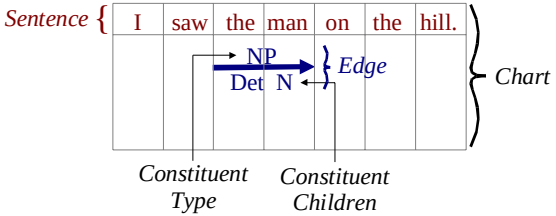


Chart Parsing: What is it?

A flexible and efficient parsing algorithm

- **Build parse trees using a *context free grammar*.**
- **Use a *chart* to record hypotheses about possible syntactic constituents. Charts contain *edges*.**
- **Each *edge* represents a possible constituent.**



Edges

Edges can represent *partial* constituents.

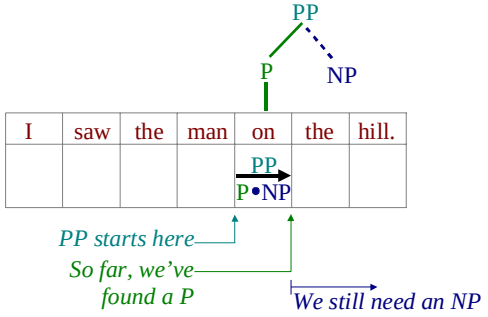
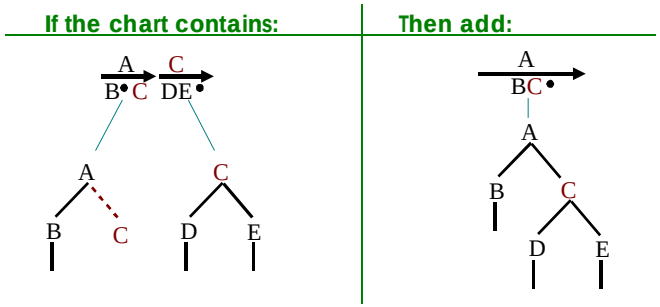


Chart Parser Rules

- Chart parser rules add new edges to the chart.
- Example – The fundamental rule:



Demo – ChartParser

NLTK Contents

NLTK provides everything necessary to learn about natural language processing tasks.

- Python Modules define basic data types, standard interfaces, and sample implementations for each task.
- Tutorials provide a gentle introduction to each task.
- Problem Sets provide practical experience.
- Reference Documentation give precise explanations for each component in the toolkit.
- Technical Documentation explain the toolkit's design and implementation.

Python Modules

- Define standard data types to represent structures used in natural language processing
- Define a standard interface for each NLP task
  - Standard interfaces precisely define each task
  - Standard interfaces allow tasks to be easily combined to form larger NLP systems
- Provide implementations for each NLP task interface

Current NLTK Modules

- Basics
  - Basic data types: tokens, trees, etc.
  - Basic processing techniques: tokenizing, stemming, etc.
  - Probability Modeling
- Tagging
- Parsing
  - Chart parsing
  - Chunk parsing
- Text Classification
- Visualization

Extending NLTK

- NLTK was designed to be extensible
  - There are very few dependencies between modules.
  - Allows many developers to work on the toolkit concurrently.
- There are many ways NLTK can be extended
  - machine translation
  - word sense disambiguation
  - information retrieval
  - question answering
  - entity extraction
  - etc.
- This extensibility provides excellent opportunities for student projects.

Natural Language Toolkit

If you are interested in contributing to NLTK, or have any ideas for improvements, please contact us.

URL: http://nltk.sf.net
Email: edloper@gradient.cis.upenn.edu
sb@unagi.cis.upenn.edu