Special Topics in Computer Science

# NLP in a Nutshell

**CS492B Spring Semester 2009**

Jong C. Park

Computer Science Department

Korea Advanced Institute of Science and Technology

# CORPUS PROCESSING TOOLS

# Corpora

- Types of Corpora
  - Primer: The need for balancing a corpus
  - Balanced corpora
    - LDC: The Linguistic Data Consortium
    - ELRA: The European Language Resources Association
  - Annotated corpora
    - Each word is labeled with a linguistic tag such as a part of speech or a semantic category.
    - Cf. Treebanks: collections of parse trees (or syntactic structures of sentences)
  - Spoken corpora

# Corpora

| | English | French | German |
|---|---|---|---|
| **Most frequent words in a collection of contemporary running texts** | the | de | der |
| | of | le (article) | die |
| | to | la (article) | und |
| | in | et | in |
| | and | les | des |
| **Most frequent words in the Genesis** | and | et | und |
| | the | de | die |
| | of | la | der |
| | his | à | da |
| | he | il | er |

# Corpora

- Corpora and Lexicon Building
  - Lexicography
    - The science of building lexicons and writing dictionaries
    - Cf. citations
  - Concordance
    - Consists of text excerpts centered on a specific word and surrounded by a limited number of words before and after it

# Corpora

| Language | Concordances |
|---|---|
| English | s beginning of miracles did Je |
| | n they saw the miracles which |
| | n can do these miracles that t |
| | ain the second miracle that Je |
| | e they saw his miracles which |
| French | le premier des miracles que fi |
| | i dirent: Quel miracle nous mo |
| | om, voyant les miracles qu'il |
| | peut faire ces miracles que tu |
| | s ne voyez des miracles et des |

# Corpora

- Concordancing
  - A powerful tool to study usage patterns and to write definitions.
  - Provides evidence on certain preferences between verbs and prepositions, adjectives and nouns, recurring expressions, or common syntactic forms.
    - Cf. collocations

# Corpora

| | **English** | **French** | **German** |
|---|---|---|---|
| You say | *Strong tea* | *Thé fort* | *Kräftiger Tee* |
| | *Powerful computer* | *Ordinateur puissant* | *Starker Computer* |
| You don't say | *Strong computer* | *Thé puissant* | *Starker Tee* |
| | *Powerful tea* | *Ordinateur fort* | *Kräftiger Computer* |

| **Preference for *strong* over *powerful*** | | | **Preference for *powerful* over *strong*** | | |
|---|---|---|---|---|---|
| strong w | powerful w | w | strong w | powerful w | w |
| 161 | 0 | showing | 1 | 32 | than |
| 175 | 2 | support | 1 | 32 | figure |
| 106 | 0 | defense | 3 | 31 | minority |
| … | | | | | |

# Corpora

- Corpora as Knowledge Sources
  - Short term:
    - Describe usage more accurately
    - Assess tools: part-of-speech taggers, parsers.
    - Learn statistical/machine learning models for speech recognition, taggers, parsers
    - Derive automatically symbolic rules from annotated corpora
  - Longer term:
    - Semantic processing
    - Texts are the main repository of human knowledge

# Finite-State Automata

- A Description
  - A flexible to tool to search and process text
  - A FSA accepts and generates strings, such as *ac, abc, abbc, abbbc, abbbbbbbbbbbbc, etc.*
  - Fig. 2.1. A finite-state automaton
  - Fig. 2.2. A finite-state automaton with an ε-transition

# Finite-State Automata

- Mathematical Definition
  - *Q: a finite set of states.*
  - Σ: a finite set of symbols or characters (or the input alphabet).
  - *q0: the start state.*
  - *F: the set of final states,* $F \subseteq Q$.
  - δ: the transition function $Q \times \Sigma \rightarrow Q$ where $\delta(q, i)$ *returns the state* to which the automaton moves when it is in state *q and* consumes the input symbol *i.*

# Finite-State Automata

- Finite-State Automata in Prolog

```
% The start state
start(q0).
% The final states
final(q2).
% The transitions
transition(q0, a, q1).
transition(q1, b, q1).
transition(q1, c, q2).
```

```
accept(Symbols) :-
    start(StartState),
    accept(Symbols, StartState).
accept([], State) :-
    final(State).
accept([Symbol | Symbols], State) :-
    transition(State, Symbol,
          NextState),
    accept(Symbols, NextState).
```

# Finite-State Automata

- Deterministic and Nondeterministic Automata
- Building a Deterministic Automata from a Nondeterministic One
- Searching a String with a Finite-State Automaton

# Finite-State Automata

- Operations on Finite-State Automata

- Resources
  - The FSA Utilities
    http://odur.let.rug.nl/~vannoord/Fsa
  - The FSM Library
    http://www.research.att.com/sw/tools/fsm

# Regular Expressions

- Repetition Metacharacters
- The Longest Match
- Character Classes
- Nonprintable Symbols or Positions

# Regular Expressions

- Union and Boolean Operators
- Operator Combination and Precedence

# Programming with Regular Expressions

- Perl

- Matching
  - If ($line =~ m/ab*c/)
- Substitutions
  - $line =~ s/regex/replacement/g
- Translating Characters
  - $line =~ tr/A-Z/a-z/;

# Programming with Regular Expressions

- String Operators
  - Concatenate (.) and compare strings (ge, le, gt, lt, eq, ne)
- Back References
  - s/(.)\1\1/***/g

# Finding Concordances

- Concordances in Prolog

```
concordance(Pattern, List, Span, Line) :-
    name(Pattern, Lpattern),
    prepend(Lpattern, Span, LeftPattern),
    append(_, Rest, List),
    append(LeftPattern, End, Rest),
    prefix(End, Span, Suffix),
    append(LeftPattern, Suffix, LLine),
    name(Line, LLine).
```

# Finding Concordances

- Concordances in Perl

# Approximate String Matching

- Edit Operations
  - Transform a source string into a target string
    - Examples: Copy, Substitution, Insertion, Deletion, Reversal (or Transposition)
- Minimum Edit Distance
  - the operation sequence that has the minimal cost needed to transform the source string into the target string

# Approximate String Matching

- Searching Edits in Prolog

```
edit_distance(Source, Target, Edits, Cost) :-
    edit_distance(Source, Target, Edits, 0, Cost).
```

# PROGRAMMING ASSIGNMENT

# Programming Assignment 1

- Implement a prototype dictionary look-up program in Prolog with the following characteristics:
  - The input is an English word, which may or may not appear in the dictionary.
  - If the input is listed in the dictionary, the program returns the content field of the corresponding dictionary entry.
  - Otherwise, the program finds an alternative entry in the dictionary with the minimum edit distance from the input word, and returns this word to ask the user if this is what is desired.
- You may design your own dictionary with entries over 100 words.

# Programming Assignment 1

- Mail to cs492@nlp.kaist.ac.kr, with
  - Your dictionary look-up program
  - Your dictionary
  - A report explaining the structure, algorithm, and usage of your program
- Due
  - February 24, 2009 (1:00 pm)

# Programming Assignment 1

- Evaluation : 350 points
  - 50 points for completing the assignment
  - 100 points for the report
  - 50 points for the dictionary look-up function
  - 150 points for the suggestion function

- Late Submission
  - We will deduct 10% of the perfect score for one day late submission, 20% of the perfect score for two days late submission.
  - You will get no score if you delay more than three days.