Regular Expressions and Automata

Lecture #2
This lecture comes from Kathleen F. McCoy at Univ. of Delaware

Natural Language Understanding

• Associate each input (acoustic signal/character string) with a meaning representation.

• Carried out by a series of components:
  – Each component acts as a translator from one representation to another
  – In general, each component adds successively ‘richer’ information to the output
Representations and Algorithms for NLP

- Representations: formal models used to capture linguistic knowledge
- Algorithms manipulate representations to analyze or generate linguistic phenomena
- Simplest often produce best performance but….the 80/20 Rule and “low-hanging fruit”
NLP Representations

• State Machines
  – FSAs, FSTs, HMMs, ATNs, RTNs
• Rule Systems
  – CFGs, Unification Grammars, Probabilistic CFGs
• Logic-based Formalisms
  – 1st Order Predicate Calculus, Temporal and other Higher Order Logics
• Models of Uncertainty
  – Bayesian Probability Theory

NLP Algorithms

• Most are parsers or transducers: accept or reject input, and construct new structure from input
  – State space search
    • Pair a partial structure with a part of the input
    • Spaces too big and ‘best’ is hard to define
  – Dynamic programming
    • Avoid recomputing structures that are common to multiple solutions
Regular Expressions

• Can be viewed as a way to specify:
  – Search patterns over text string
  – Design of a particular kind of machine, called a Finite State Automaton (FSA)

• These are really equivalent

Uses of Regular Expressions in NLP

• As grep, perl: Simple but powerful tools for large corpus analysis and ‘shallow’ processing
  – What word is most likely to begin a sentence?
  – What word is most likely to begin a question?
  – In your own email, are you more or less polite than the people you correspond with?

• With other unix tools, allow us to
  – Obtain word frequency and co-occurrence statistics
  – Build simple interactive applications (e.g. Eliza)

• Regular expressions define regular languages or sets
Regular Expressions

- **Regular Expression**: Formula in algebraic notation for specifying a set of strings

- **String**: Any sequence of alphanumerical characters
  - Letters, numbers, spaces, tabs, punctuation marks

- **Regular Expression Search**
  - Pattern: specifying the set of strings we want to search for
  - Corpus: the texts we want to search through

### Some Examples

<table>
<thead>
<tr>
<th>RE</th>
<th>Description</th>
<th>Uses</th>
</tr>
</thead>
<tbody>
<tr>
<td>./</td>
<td>Wild card; Any char</td>
<td>A blank line?</td>
</tr>
<tr>
<td>/a/</td>
<td>Any ‘a’</td>
<td>Line with words?</td>
</tr>
<tr>
<td>/[ab]/</td>
<td>A choice</td>
<td>Rhyming words?</td>
</tr>
<tr>
<td>/[a-z]/</td>
<td>l.c. char (range)</td>
<td>Common noun?</td>
</tr>
<tr>
<td>/[A-Z]/</td>
<td>u.c. char</td>
<td>Proper noun?</td>
</tr>
<tr>
<td>/[^?!.]/</td>
<td>Neg of set</td>
<td>Not S-final punc</td>
</tr>
<tr>
<td>RE</td>
<td>Description</td>
<td>Uses?</td>
</tr>
<tr>
<td>----------------</td>
<td>---------------------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td><code>/a*</code></td>
<td>Zero or more a's</td>
<td>Optional doubled modifiers (words)</td>
</tr>
<tr>
<td><code>/a+</code></td>
<td>One or more a's</td>
<td>Non-optional...</td>
</tr>
<tr>
<td><code>/a?</code></td>
<td>Zero or one a's</td>
<td>Optional...</td>
</tr>
<tr>
<td>`/cat</td>
<td>dog/`</td>
<td>'cat' or 'dog'</td>
</tr>
<tr>
<td><code>/^cat\.$/</code></td>
<td>A line that contains only cat. *anchors beginning, $ anchors end of line.</td>
<td>??</td>
</tr>
<tr>
<td><code>\bun\B/</code></td>
<td>Beginnings of longer strings</td>
<td>Words prefixed by 'un'</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RE</th>
<th>E.G.</th>
</tr>
</thead>
<tbody>
<tr>
<td>`/pupp(y</td>
<td>ies)/`</td>
</tr>
<tr>
<td><code>/ (.+ier and \1ier /</code></td>
<td>happier and happier, fuzzier and fuzzier</td>
</tr>
</tbody>
</table>
Optionality and Repetition

- /\[Ww\]oodchucks\?\// matches woodchucks, Woodchucks, woodchuck, Woodchuck
- /colou?r/ matches color or colour
- /he\{3\}/ matches heee
- /\(he\)\{3\}/ matches hehehe
- /\(he\)\{3,\}/ matches a sequence of at least 3 he's

Operator Precedence Hierarchy

1. Parentheses       ()
2. Counters          * + ? {}
3. Sequence of Anchors   ^my end$
4. Disjunction        |

Examples
/moo+/ 
/try|lies/ 
/and|or/
A Simple Exercise

• Write a regular expression to find all instances of the determiner “the”:

  /the/  
  /\[tT\]he/  
  /\b\[tT\]he\b/  
  /(^|[^a-zA-Z]\[tT\]he[^a-zA-Z]/

The recent attempt by the police to retain their current rates of pay has not gathered much favor with the southern factions.

A Simple Exercise

• Write a regular expression to find all instances of the determiner “the”:

  /\the/  

The recent attempt by the police to retain their current rates of pay has not gathered much favor with southern factions.
A Simple Exercise

• Write a regular expression to find all instances of the determiner “the”:
  /the/
  /\[tT\]he/  

The recent attempt by the police to retain their current rates of pay has not gathered much favor with the southern factions.
A Simple Exercise

• Write a regular expression to find all instances of the determiner “the”:
  /the/
  /[tT]he/  
  /\b[tT]he\b/  
  /(^|[^a-zA-Z])[tT]he[^a-zA-Z]/

The recent attempt by the police to retain their current rates of pay has not gathered much favor with the southern factions.
The Two Kinds of Errors

- The process we just went through was based on fixing errors in the regular expression
  - Errors where some of the instances were missed (judged to not be instances when they should have been) – False negatives
  - Errors where the instances were included (when they should not have been) – False positives

- This is pretty much going to be the story of the rest of the course!

Substitutions (Transductions)

- Sed or ‘s’ operator in Perl
  - s/regexp/pattern/
  - s/I am feeling (.++)/You are feeling \1?/
  - s/I gave (.+) to (.+)/Why would you give \2 \1?/
Examples

• Predictions from a news corpus:
  – Which candidate for Governor of California is mentioned most often in the news? Is going to win?
  – What stock should you buy?
  – Which White House advisers have the most power?

• Language use:
  – Which form of comparative is more frequent: "oftener" or "more often"?
  – Which pronouns are conjoined most often?
  – How often do sentences end with infinitival "to"?
  – What words most often begin and end sentences?
  – What’s the most common word in your email? Is it different from your neighbor?

• Personality profiling:
  – Are you more or less polite than the people you correspond with?
  – With labeled data, which words signal friendly messages vs. unfriendly ones?
Finite State Automata

- Regular Expressions (REs) can be viewed as a way to describe machines called Finite State Automata (FSA, also known as automata, finite automata).

- FSAs and their close variants are a theoretical foundation of much of the field of NLP.

![Finite State Automata Diagram]

- FSAs recognize the regular languages represented by regular expressions
  - SheepTalk: /baa+/!

- Directed graph with labeled nodes and arc transitions
  - Five states: q0 the start state, q4 the final state, 5 transitions
Formally

- FSA is a 5-tuple consisting of
  - \( Q \): set of states \( \{q_0,q_1,q_2,q_3,q_4\} \)
  - \( \Sigma \): an alphabet of symbols \( \{a,b,!\} \)
  - \( q_0 \): a start state
  - \( F \): a set of final states in \( Q \) (\( q_4 \))
  - \( \delta(q,i) \): a transition function mapping \( Q \times \Sigma \) to \( Q \)

```
\begin{tikzpicture}
  \node (q0) at (0,0) {q_0};
  \node (q1) at (1,0) {q_1};
  \node (q2) at (2,0) {q_2};
  \node (q3) at (3,0) {q_3};
  \node (q4) at (4,0) {q_4};

  \draw[->] (q0) -- node[above] {b} (q1);
  \draw[->] (q1) -- node[above] {a} (q2);
  \draw[->] (q2) -- node[above] {a} (q3);
  \draw[->] (q3) -- node[above] {!} (q4);
\end{tikzpicture}
```

State Transition Table for SheepTalk

<table>
<thead>
<tr>
<th>State</th>
<th>Input</th>
<th>b</th>
<th>a</th>
<th>!</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>1</td>
<td>∅</td>
<td>2</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>2</td>
<td>∅</td>
<td>3</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>3</td>
<td>∅</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
</tr>
</tbody>
</table>
Recognition

- Recognition (or acceptance) is the process of determining whether or not a given input should be accepted by a given machine.

- In terms of REs, it’s the process of determining whether or not a given input matches a particular regular expression.

- Traditionally, recognition is viewed as processing an input written on a tape consisting of cells containing elements from the alphabet.

- FSA recognizes (accepts) strings of a regular language
  - baa!
  - baaa!
  - baaa!
  - ...

- Tape metaphor: a rejected input
D-Recognize

Function D-Recognize(tape, machine) returns accept or reject
Index ← Beginning of tape
Current-state ← Initial state of the machine
loop
If end of input has been reached then
    If current-state is an accept state then
        return accept
    Else
        return reject
elseif transition-table[current-state, tape[index]] is empty then
    return reject
else
    Current-state ← transition-table[current-state, tape[index]]
    Index ← index + 1
end

<table>
<thead>
<tr>
<th>State</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>b</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>Ø</td>
</tr>
<tr>
<td>2</td>
<td>Ø</td>
</tr>
<tr>
<td>3</td>
<td>Ø</td>
</tr>
<tr>
<td>4</td>
<td>Ø</td>
</tr>
</tbody>
</table>
Key Points About D-Recognize

- Deterministic means that the code always knows what to do at each point in the process
- Recognition code is universal for all FSAs. To change to a new formal language:
  - change the alphabet
  - change the transition table
- Searching for a string using a RE involves compiling the RE into a table and passing the table to the interpreter

Determinism and Non-Determinism

- **Deterministic**: There is at most one transition that can be taken given a current state and input symbol.

- **Non-deterministic**: There is a choice of several transitions that can be taken given a current state and input symbol. (The machine doesn't specify how to make the choice.)
Non-Deterministic FSAs for SheepTalk

FSAs as Grammars for Natural Language

Can you use a regexpr to capture this too?
Problems of Non-Determinism

- ‘Natural’….but at any choice point, we may follow the wrong arc
- Potential solutions:
  - Save backup states at each choice point
  - Look-ahead in the input before making choice
  - Pursue alternatives in parallel
  - Determinize our NFSAs (and then minimize)
- FSAs can be useful tools for recognizing – and generating – subsets of natural language
  - But they cannot represent all NL phenomena (Center Embedding: The mouse the cat ... chased died.)

Recognition as Search

- Systematically Searching for Solutions -> state-space search algorithm
- “Order” – in which the states in the space are considered
- Depth-first search or Last in First Out (LIFO) – stack
  - Figure 2.22
- Breadth-first search or First in First Out (FIFO) – queue
  - Figure 2.23
Summing Up

• Regular expressions and FSAs can represent subsets of natural language as well as regular languages
  – Both representations may be impossible for humans to understand for any real subset of a language
  – But they are very easy to use for smaller subsets

• Next time: Read Ch 3

• For fun:
  – Think of ways you might characterize features of your email using only regular expressions