Introduction to Natural Language Processing

Chapter 2

Outline

Administration - Blackboard, Readings, NLP Meeting Time, Thesis Defense on Wednesday, No Office Hours on Tuesday

Overview of Next Lectures

Regular Expressions and Finite State Automata

Deterministic Recognition

Generative Grammars and Formal Languages

Non-Determinism

Recognition as Search

Next Lectures: Words

The next set of lectures will begin our study of "words" and will cover three related topics

- finite state automata
- finite state transducers
- English morphology

First we will

- Review some of the simple representations (regular expressions and finite state automata) and ask ourselves how we might use them to do interesting and useful things
- Think about the limits of these simple approaches: when do we need more?

Regular Expressions

Useful ways to view Regular Expressions

- as a practical way to specify textual search strings
- as a way to specify the design of a particular kind of machine

As we will see, these are equivalent

Uses of Regular Expressions in NLP

Simple but powerful tools such as grep and Perl allow large corpus analysis and "shallow" processing

- what word is most likely to begin a sentence vs. a question?
- which pronouns are conjoined most often?
- in your own email, are you more or less polite than the people you correspond with (and with labeled data, how can you learn this)?
- which candidate for President is mentioned most often in the news (and with labeled data, which is mentioned most favorably)?

With other tools, we can also

- obtain word frequency and co-occurrence statistics
- build simple interactive applications like Eliza

Regular Expressions: Text Searches

Everybody does it

- Perl, emacs, Word, sed, awk, grep, netscape, etc.

Basic regular expression patterns (Perl notation) - J&M pages 23-27

- sequence (slashes)
- disjunction (brackets)
- range (brackets plus dash)
- negation (open bracket plus caret)
- optionality (question-mark)
- Kleene star (asterisk)
- wildcard (period)
- anchors (caret, dollar-sign, \b, \B)

Brackets and Character Disjunction

RE	Match	Example Patterns
`/[wW]oodchuck/`	Woodchuck or woodchuck	"<u>Woodchuck</u>"
`/[abc]/`	'a', 'b', *or* 'c'	"In uomini, in sold<u>a</u>ti"
`/[1234567890]/`	any digit	"plenty of <u>7</u> to 5"

A Simple Example

Finding instances of the determiner "the" (using a <u>pattern</u>, over a <u>corpus</u>)

```
/the/
/[tT]he/
/\b[tT]h\b/
```

The second RE catches "the" when it begins a sentence, while the third RE does not match "the" when embedded in another word such as "other"

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 be not instances when they should have been)
- errors where the instances were included (when they should not have been)

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

REs: Text Searches (continued)

Disjunction, grouping, and precedence - J&M pages 27-28

Operators are greedy

Advanced operators - J&M pages 30-31

Substitution and memory - J&M pages 31-33

ELIZA: Substitutions Using Memory

```
User: Men are all alike.

s/.* all .*/IN WHAT WAY/

ELIZA: IN WHAT WAY

User: They're always bugging us about something or other.

s/.* always .*/CAN YOU THINK OF A SPECIFIC EXAMPLE/

ELIZA: CAN YOU THINK OF A SPECIFIC EXAMPLE

User: My boyfriend says I'm depressed.

s/.* I'm (depressed|sad) .*/I AM SORRY TO HEAR YOU ARE \1/

ELIZA: I AM SORRY TO HEAR YOU ARE DEPRESSED
```

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.

FSAs as Directed Graphs

A sheep language

- baa!
- baaa!
- ...

A regular expression for the sheep language

- /baa+!/

A graphical view of the sheep language: Figure 2.10

- five states (nodes in the graph)
- q0 is the start state (incoming arrow)
- q4 is the final state (double circle)
- five transitions (arcs in the graph)
- deterministic

Alternatives for our Sheep Language

/baa+!/

/baaa*!/

More on this later...

Outline

Last Class

- Overview of Next Lectures
- Regular Expressions and Finite State Automata

Today's Class

- Readings Due; Thesis Defense Reminder
- Finite State Automata
- Deterministic Recognition
- Review REs, FSAs, and Deterministic Recognition
- Generative Grammars and Formal Languages
- Non-Determinism
- Recognition as Search
- First Homework

A More Formal View

FSAs can be formally specied as a 5-tuple

- a finite set of N states $q_0, q_1 \ldots q_n$: Q
- a finite input alphabet of symbols: Σ
- the start state: q_0
- the set of final states: F $\subseteq$ Q
- the transition function that maps Q $\times$ Σ to Q

A Note on Alphabets

You shouldn't view the term alphabet too narrowly. In particular, you don't have to limit it to letters. Any kind of symbol will do.

Furthermore, those symbols can stand for objects that themselves have internal structure.

Simple Dollars and Cents Example

Figure 2.17

- alphabet of words
- accepts phrases like *ten cents, three dollars, twenty dollars thirty five cents*
- but...
  - accepts *one dollars*
  - doesn't accept *hundred dollars*

Yet Another Representation

FSAs can also be encoded as *State Transition Tables*

- Q $= \{q_0, q_1, q_2, q_3, q_4\}$
- $\Sigma = \{$a,b,!$\}$
- F=$\{q_4\}$
- transition function =

	Input		
State	b	a	!
0	1	∅	∅
1	∅	2	∅
2	∅	3	∅
3	∅	3	4
4:	∅	∅	∅

Adding a failing state (Figure 2.15)

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, its 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

Figure 2.11

D-Recognize

```
function D-RECOGNIZE(tape, machine) returns accept or reject

  index ← Beginning of tape
  current-state ← Initial state of machine
  loop
    if End of input has been reached then
      if current-state is an accept state then
        return accept
      else
        return reject
    elsif transition-table[current-state,tape[index]] is empty then
      return reject
    else
      current-state ← transition-table[current-state,tape[index]]
      index ← index + 1
  end
```

D-Recognize (continued)

Key points about d-recognize

- Deterministic means that the code always knows what to do at each point in the process.
- d-recognize is essentially a crude table-driven interpreter.
- the recognition code is universal for all FSAs. To change to a new formal language, you merely change the alphabet and the table. The recognition code stays the same.
- thus, searching for a string using a RE involves compiling the RE into a table and passing the table to an interpreter.

Example Traces

baaa! (Figure 2.14)

abc

Review

REs are compact textual representations of FSAs, themselves compact graphical representations of formal languages (sets of strings) called regular languages.

Recognition is the process of determining if an input is in the language accepted/generated/represented by a given FSA.

Recognition in deterministic machines is easy (table-driven).

FSAs can be useful tools for recognizing - and generating - $subsets$ of natural language.

Review Examples

What would the machine that accepts the following dialect of sheep language (/baa+!/) look like?

- /baa{49}!/

What does the following regular expression do to the input "DOES SHE LIKE CLASS"

```
s/([A-Z]+) +([A-Z]+)/ \2 \1 /
```

Formal Languages

A Formal Language is a set of strings composed of symbols from a finite set of symbols (the alphabet).

FSAs (and regular expressions) define formal languages (without having to explicitly enumerate the set).

Given a machine m (such as a particular FSA) L(m) means the formal language characterized by m.

- L(sheeptalk FSA) $=$ $\{$baa!, baaa!, baaaa!, $\ldots\}$ (an infinite set)

Generative Grammars

The term Generative refers to the idea that FSAs can be viewed as generators of formal languages as well as acceptors. This dual view will pop up again and again.

To generate, you traverse the machine and write transition symbols on the tape rather than reading them.

Non-Deterministic FSAs (NFSAs)

DFSA

Figure 2.10

- behavior is fully *determined* by current state and input

NFSA1

Figure 2.18

- an automata with *decision points*

Non-Determinism (continued)

Yet another way of introducing non-determinism is to use ϵ-transitions (arcs with no symbols)

NFSA2:

Figure 2.19

Key Point: ϵ-transitions do not examine / advance the tape

NFSA Transition Tables

	Input			
State	`b`	`a`	`!`	ε
0	1	∅	∅	∅
1	∅	2	∅	∅
2	∅	2,3	∅	∅
3	∅	∅	4	∅
4:	∅	∅	∅	∅

- epsilon input column
- cells can be *lists* of nodes

Equivalence of D and ND Machines

Any ND machine can be converted to a D machine by a fairly simple construction (e.g., Hopcroft and Ullman 1979).

This means that formally they have the same power - they can recognize exactly the same classes of language. Non-determinism does not add formal power to FSAs.

It also means that one way of doing recognition would be to convert an ND-machine to a D-machine and then do recognition with D-Recognize.

Non-Deterministic Recognition

Back to recognition . . .

In a non-deterministic machine, there exists at least one path directed through the machine by a string in the language that leads to an accept condition.

There may exist paths driven by strings in the language that do no lead to an accept condition. Doesn't matter if some wrong choices lead to a reject condition.

In a non-deterministic machine no path directed through the machine by a string outside the language leads to an accept condition.

Non-Determinism (continued)

So... non-deterministic recognition succeeds whenever it is driven by an input to any accept condition.

However, being driven to a reject condition by an input does not imply it should be rejected.

To reject a string it has to be sure that there are no paths that can lead to an accept.

That means that the problem of non-deterministic recognition can be thought of as a search problem.

The Problem of Choice

Choice in non-deterministic models comes up again and again in NLP

Several standard solutions

- backup (search, this chapter)
  - mark input/state at choice points
  - if wrong choice, use marker to backup and try another choice
- lookahead
  - look ahead in the input to help make choice
- parallelism
  - look at all choices in parallel
- determinize the model

The Backup Approach

After a wrong choice leads to a dead-end (no input left, no legal transitions), return to a previous choice point and pursue another *unexplored* choice.

Thus, at each choice point, the search process needs to remember the <u>search state</u>:

- tape position
- FSA node (or *machine* state)

Example: Figure 2.22

Key Points

A search-state is a pairing of a single machine-state with a position on the input tape.

By keeping track of not yet explored search-states, a recognizer can systematically explore all possible paths through a machine given some input.

ND-Recognize Code

```
function ND-RECOGNIZE(tape, machine) returns accept or reject

  agenda ← {(Initial state of machine, beginning of tape)}
  current-search-state ← NEXT(agenda)
  loop
    if ACCEPT-STATE?(current-search-state) returns true then
       return accept
    else
       agenda ← agenda ∪ GENERATE-NEW-STATES(current-search-state)
    if agenda is empty then
       return reject
    else
       current-search-state ← NEXT(agenda)
  end

function GENERATE-NEW-STATES(current-state) returns a set of search-
states

  current-node ← the node the current search-state is in
  index ← the point on the tape the current search-state is looking at
  return a list of search states from transition table as follows:
    (transition-table[current-node,ε], index)
    ∪
    (transition-table[current-node, tape[index]], index + 1)

function ACCEPT-STATE?(search-state) returns true or false

  current-node ← the node search-state is in
  index ← the point on the tape search-state is looking at
  if index is at the end of the tape and current-node is an accept state of machine
then
    return true
  else
    return false
```

Why Bother

Given all of the above, non-determinism seems to be more trouble that its worth. Why introduce it at all?

- more intuitive to solve certain problems
- often lots smaller (n versus possibly 2^n)

Recognition as Search

State-Space Search: Algorithms such as ND-Recognize, which systemically search for solutions

- problem definition creates a space of possible solutions
- goal is to explore the space
  - return answer when found
  - reject when space has been exhaustively explored

ND-Recognize as State-Space Search

Search states: pairings of machine states with tape position

State-space: all possible pairings given the machine

Search goal: find an accept state/end of tape pair

Ordering the Search

Note that NEXT is undefined in ND-Recognize

Depth-first search

- most recently created states considered first
- agenda is a <u>stack</u>
- last in first out (LIFO)
- previous example trace

Breadth-first search

- states considered in creation order
- agenda is a <u>queue</u>
- first in first out (FIFO)
- instead of picking one choice and following it, examine all possible choices

Dynamic programming / A*

- will be discussed later in the course

FSAs as NL Grammars

How would we write a FSA for all the ways to refer to someone like the Reverend Dr. Martin Luther King?

How about if we wanted to extract all the proper names in the news?

Homework

Recall Policies

Look at Blackboard for details of HW1

Summing Up

Regular expressions and FSAs can represent subsets of natural language as well as regular languages

- both representations may be difficult for humans to understand for any real subset of a language
- but they are relatively easy to use for small subsets
- can be hard to scale up: when many choices at any point (e.g. surnames)
- cannot represent all NL phenomena (center embedding: The mouse the cat chased died)

For Next Time

Chapter 3