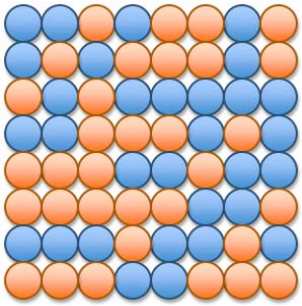


UNIT 13B

AI: Natural Language Processing

Announcement (1)

- Exam on Wednesday November 28
  - Covered topics: Randomness, Concurrency, Internet, Simulation, AI, Recursion
  - Rooms for Exam 3: Sections A, B, C, D, H, I, J, K go to Rashid Auditorium (GHC 4401). Sections E, F, G, L, M, N go to Baker Hall A51 (Giant Eagle Auditorium).
  - Usual office hours for review sessions. Locations to be announced on the course Web site and Piazza.

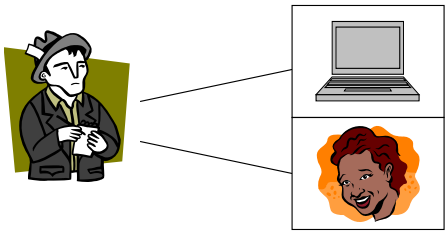
Announcement (2)

- Problem Set 12 to be released by tomorrow:
  - NOT FOR CREDIT.
  - Covers material included in the exam
  - Covers material that will be covered in class after Thanksgiving

3

The Turing Test

- Turing publishes Computing Machinery and Intelligence in 1950.
- Describes a test (now called the Turing Test) to determine whether a computer can be called intelligent.



A machine is considered intelligent if an interrogator cannot tell if a human or a computer is answering a set of questions using typewritten responses.

Natural Language Processing

- Natural language processing involves the interactions between a computer and a human via spoken (or written) language.
- ELIZA
  - One of the first programs to build a conversation between a human and a computer, created by Joseph Weizenbaum in 1966.
  - Computer responses were created based on *scripts*.
  - Most famous example was DOCTOR which simulated a Rogerian psychotherapist.

ELIZA in Popular Culture

- Human: Who would you vote for- Mitt Romney or Barack Obama?
- Siri (Apple's voice activated service): I can't vote. But if I did I would vote for ELIZA. She knows all.

ELIZA

H: My father wouldn't buy me a puppy.

C: Tell me more about your family.

H: I am sad I never got a puppy.

C: Can you explain what made you sad you never got a puppy?

H: I am sad we never had a chance to play fetch.

C: I am sorry to hear you are sad we never had a chance to play fetch.

Searching a String

```
>> s = "My dog ate the caterpillar."
=> "My dog ate the caterpillar."
>> s.include?("dog")
=> true
>> s.include?("Caterpillar")
=> false
>> s.include?("cat")
=> true
```

Regular Expressions

- A regular expression is a rule that describes the format required for a specific string.

```
>> s = “how now browner cow ow”
=> “how now browner cow ow”
>> r = /.ow/
=> /.ow/
```

r is a regular expression that says match any string consisting of 3 characters, where the first character is anything and the next 2 characters are ‘o’ and ‘w’ exactly

Iteration with a Regular Expression

- Scan scans through the string looking for anything that matches the regular expression passed to it.

```
>> s = “how now browner cow ow”
=> “how now browner cow ow”
>> r = /.ow/
=> /.ow/
>> s.scan(r)
=> [“how”, “now”, “row”, “cow”, “ ow”]
```

Examples

Syntax	Semantics
`.`	`Match any single character`
`.*`	`Match any number of characters`
`x | y | z`	`Match x or y or z`

Specific syntax and semantics may vary depending on the programming language, implementation, or library.

11

Regular Expressions in Practice

- Editors
  - Searching/replacing text
  - Syntax highlighting
- Query languages

RubyLabs: Pattern

include ElizaLab

- A sentence **Pattern** is a mapping from a regular expression to a set of 1 or more responses.
- Example:

creates a regular expression based on the first argument

```
>> p1 = Pattern.new("dog",
   ["Tell me more about your pet",
       "Go on"]
=> dog: ["Tell me more about your
pet", "Go on"]
```


More about Patterns

- The **apply** method tries to match an input sentence to a regular expression. If it can, it returns one of supplied response strings.

```
>> p1.apply("I love my dog.")
=> "Tell me more about your pet."
>> p1.apply("My dog is really smart.")
=> "Go on."
>> p1.apply("Much smarter than my cat.")
=> nil
```

```
p1 = Pattern.new("dog",
      ["Tell me more about your pet","Go on"]
)
```

Groups

- We can specify a **group** so that any member will cause a match during a scan.

```
>> p2 = Pattern.new("(cat|dog|bird)",
   ["Tell me more about your pet", "Go on"]
>> p2.apply("My dog is smelly.")
=> "Go on."
>> p2.apply("My cat ate my bird.")
=> "Tell me more about your pet."
>> p2.apply("I miss Polly a lot.")
=> nil
```


Placeholders

- We can use placeholders to store the part of a pattern that matches so we can use it in the response.

```
>> p = Pattern.new("(cat|dog|bird)")
>> p.add_response("Tell me more about the $1")
>> p.add_response("A $1? Interesting.")
>> p.apply("A dog ate my homework.")
=> "Tell me more about the dog."
>> p.apply("My cat ate my bird.")
=> "A cat? Interesting."
```

Placeholders (cont'd)

```
>> p = Pattern.new("I (like|love|hate) my
  (cat|dog|bird)"
>> p.add_response("Why do you $1 your $2?")
>> p.add_response("Tell me more about your $2")
>> p.apply("I like my dog.")
=> "Why do you like your dog?"
>> p.apply("I hate my cat.")
=> "Tell me more about your cat."
```


Wildcards

- We can use a wildcard symbol (.*) to match any number of characters.

```
>> p = Pattern.new("I am afraid of (.*)")
>> p.add_response("Why are you afraid of $1?")
>> p.apply("I am afraid of ghosts")
=> "Why are you afraid of ghosts?"
>> p.apply("I am afraid of 15110")
=> "Why are you afraid of 15110?"
```

A Note About **Pattern**

- Pattern takes a string and converts it to a regular expression, adding some special characters.

```
>> p = Pattern.new(“dog”)
>> p.regexp            => /\bdog\b/i
>> p = Pattern.new(“I like .*”)
>> p.regexp            => /\bI like (.*)/i
>> p = Pattern.new(“.eat”)
>> p.regexp            => /.eat\b/i
```

\b word boundary
i ignore case

Postprocessing

- To make things more realistic, we can replace personal pronouns with their opposites.

```
>> p = Pattern.new(“I am (.*)”,
        [“Are you really $1?”])
>> p.apply(“I am near my car”)
=> “Are you really near my car?”
>> p.apply(“I am annoyed at you”)
=> “Are you really annoyed at you?”
```

Postprocessing (cont'd)

```
>> Eliza.post["my"] = "your"
>> Eliza.post["you"] = "me"
>> Eliza.post
=> {"my" => "your", "you" => "me" }
>> p.apply("I am near my car.")
=> "Are you really near your car?"
>> p.apply("I am annoyed at you.")
=> "Are you really annoyed at me?"
>> p.apply("I am sad, my my my.")
=> "Are you really sad, your your your?"
```

An associative array like we used in Huffman trees.

Preprocessing

- Preprocessing is used to transform part of a sentence before pattern matching is performed.

```
>> p = Pattern.new("I am afraid of (.*)")
>> p.add_response("Why are you afraid of $1?")
>> p.apply("I'm afraid of ghosts")
=> nil
>> Eliza.pre["I'm"] = "I am"
>> p.apply("I'm afraid of ghosts")
=> "Why are you afraid of ghosts?"
```

Algorithm for a Conversation

1. Perform any preprocessing on the sentence.
2. Break the sentence into words.
3. If there is a rule for a word, add the rule to a priority queue.
4. Try the rules in order of decreasing priority.
5. If a rule applies to the input sentence, apply postprocessing rules to placeholder variables and return the response.

Priority Queues (Reminder)

- A priority queue (PQ) is like an array that is sorted.

```
pq = PriorityQueue.new
=> []
```

- To add element into the priority queue in its correct position, we use the **<<** operator:

```
pq << "peach"
pq << "apple"
pq << "banana"
=> ["apple", "banana", "peach"]
```

Priority Queues (cont'd)

- To remove the first element from the priority queue, we will use the **`shift`** method:

```
fruit1 = pq.shift
=> "apple"
pq
=> ["banana", "peach"]
fruit2 = pq.shift
=> "banana"
pq
=> ["peach"]
```


Eliza and the Turing Test

- Human (natural) language is very complex.

  "You said I wouldn't be afraid of horses if I visited a state fair, but when I was there I was chased by a pig, and now I'm having nightmares about bacon."

  Natural language processors (NLPs) must use **parsers** to break a sentence down into its constituent parts (noun, verb, etc.) to determine an appropriate response.

Eliza and the Turing Test

- Human (natural) language is very complex.

  "The lions were lying down in the grass."
  A: Go on.

  "They were quiet"
  A: Why were the lions quiet?

  Natural language processors (NLPs) must have **memory** to associate words (e.g. "they") to words from previous sentences.

Eliza and the Turing Test

- Human (natural) language is very complex.

  "The elephant was huge."
  A: How big was it?

  "The cat fell into the well."
  A: Was it able to get out?

  NLPs must use **semantics** to assign meaning to certain phrases in order to generate appropriate responses.

Eliza and the Turing Test

- Human (natural) language is very complex.

  "The dog ran across the backyard while I waited on my patio."
  A: What happened when the dog reached the patio?

  NLPs must establish **inferences** to add information to sentences that is not given but is assumed.

Conclusions

- Predictions of Turing and others that natural language processing would be well-established by 2000 were optimistic.
- Some notable achievements: parsing, NLP interfaces such as in airline reservations, banks etc.
- Corpus linguistics and machine learning hold promise.