ILLINOIS INSTITUTE OF TECHNOLOGY

Words and Pattern Matching

CS-585

Natural Language Processing

Derrick Higgins

Regular expressions

- A formal language for specifying text strings
- How can we search for any of these?
  - woodchuck
  - woodchucks
  - Woodchuck
  - Woodchucks

Regular Expressions: Disjunctions

- Letters inside square brackets []

Pattern	Matches
`[wW]oodchuck`	Woodchuck, woodchuck
`[1234567890]`	Any digit

- Ranges `[A-Z]`

Pattern	Matches	
`[A-Z]`	An upper case letter	`Drenched Blossoms`
`[a-z]`	A lower case letter	`my beans were impatient`
`[0-9]`	A single digit	`Chapter 1: Down the Rabbit Hole`

Negation in Disjunction

- Negations `[^Ss]`
  - Caret means negation only when first in []

Pattern	Matches	
`[^A-Z]`	Not an upper case letter	`Oyfn pripetchik`
`[^Ss]`	Neither 'S' nor 's'	`I have no exquisite reason"`
`[^e^]`	Neither e nor ^	`Look here`
`a^b`	The pattern a carat b	`Look up a^b now`

More Disjunction

- Woodchuck is another name for groundhog!
- The pipe | for disjunction

Pattern	Matches
groundhog|woodchuck	groundhog woodchuck
yours|mine	yours mine
a|b|c	= [abc]
[gG]roundhog|[Ww]oodchuck	…

? * + .

Pattern		Matches
colou?r	Optional previous char	color colour
o*h!	0 or more of previous char	h! oh! ooh! oooh! ooooh!
o+h!	1 or more of previous char	oh! ooh! oooh! ooooh!
baa+		baa baaa baaaa baaaaa
beg.n	Any char	begin begun begun beg3n

Anchors ^ $

Pattern	Matches	
`^[A-Z]`	Palo Alto	Start of string
`^[^A-Za-z]`	1 “Hello”	
`\.$`	The end.	End of string
`.$`	The end? The end!	

Character classes

Pattern	Matches
`\s`	A whitespace character
`\S`	A non-whitespace character
`\d`	A digit (`[0-9]`)
`\D`	A non-digit
`\w`	A "word" character (`[0-9a-zA-Z_]`)
`\W`	A non-word character
`[:upper:]`	An upper-case letter
`[:lower:]`	A lower-case letter

Backreferences (...) ...\n

- Sometimes we want to know which part of the text matched a part of a pattern
- We can even use it within the pattern itself, by "capturing" it in parentheses

Pattern	Matches	
`(\d)[a-z]\1`	zsdfg1a1z213	A letter bracketed by the same number on each side
`^(\d)(\d).*\2\1$`	13awdfgasdf31	A line starting with two digits, and ending with those two digits in reverse order

Example

- Find me all instances of the word "the" in a text.

  `the`

  Misses capitalized examples

  `[tT]he`

  Incorrectly returns o<u>`the`</u>`r` or <u>`the`</u>`ology`

  `\b[tT]he\b`

Regular expressions in Unix

- Utilities using regexes:
  - `grep`: search within text files for patterns
  - `sed`: programmatically edit text streams
  - `awk`: programmatically edit text streams (more complex syntax with control flow)
  - `perl`: scripting language tailored to text-munging use cases
- General rule: use `–E` flag with grep, sed in order to use "extended" regular expressions with character classes, capturing groups, etc.

Regular expressions in Python

```
import re

# Determine if match found anywhere in string
match = re.search(r“\d+”, “abc123xyz”)

# Determine if start of string matches
# Same as re.search() with a pattern anchored by “^”
match = re.match(r“\d+”, “123xyz”)

# Replace all occurrences within string
match = re.sub(r“(\d+)”, r”x\1x”, “abc123xyz”)
```

Regular expressions in Python

- Python "raw" strings come in handy for regexes

```
# This:
match = re.search(r"\S+\d+\S+", "abc123xyz")

# …is the same as this:
match = re.search("\\S+\\d+\\S+", "abc123xyz")
```

- Access captured elements

```
match = re.search(r"\S+(\d+)\S+", "abc123xyz")
match.group(1)

# Equals "123"
```

Errors

- The process we just went through was based on fixing two kinds of errors
  - Matching strings that we should not have matched (there, then, other)
    - False positives (Type I)
  - Not matching things that we should have matched (The)
    - False negatives (Type II)

Errors cont.

- In NLP we are always dealing with these kinds of errors.
- Reducing the error rate for an application often involves two antagonistic efforts:
  - Increasing accuracy or <u>precision</u> (minimizing false positives)
  - Increasing coverage or <u>recall</u> (minimizing false negatives).

Summary

- Regular expressions are surprisingly important
  - Often the first model for any text processing
- For many tasks, we use machine learning
  - But regular expressions are used as features in the classifiers
  - Can be very useful in capturing generalizations

TOKENIZATION

Text Normalization

- Every NLP task needs to do text normalization:
  1. Segmenting/tokenizing words in running text
  2. Normalizing word formats
  3. Segmenting sentences in running text

How many words?

- I do uh main- mainly business data processing
  - Fragments, corrections, filled pauses
- Seuss's cat in the hat is different from other cats!
  - **Lemma**: same stem, part of speech, rough word sense
    - cat and cats = same lemma
  - **Wordform**: the full inflected surface form
    - cat and cats = different wordforms

How many words?

they lay back on the San Francisco grass and looked at the stars and their

- **Type**: an element of the vocabulary.
- **Token**: an instance of that type in running text.
- How many?
  - 15 tokens
  - 13 types

How many words?

N = number of tokens

Church and Gale (1990): $|V| > O(\sqrt{N})$

V = vocabulary = set of types

$|V|$ is the size of the vocabulary

	Tokens = N	Types = |V|
Switchboard phone conversations	2.4 million	20 thousand
Shakespeare	884,000	31 thousand
Google N-grams	1 trillion	13 million

Simple Tokenization in UNIX

- (Inspired by Ken Church's UNIX for Poets.)
- Given a text file, output the word tokens and their frequencies

```
tr -sc 'A-Za-z' '\n' < shakes.txt
      | sort
      | uniq –c
```

Change all non-alpha to newlines

Sort in alphabetical order

Merge and count each type

```
1945 A
  72 AARON
  19 ABBESS
   5 ABBOT
 ... ...
```

```
25 Aaron
 6 Abate
 1 Abates
 5 Abbess
 6 Abbey
 3 Abbot
…. …
```

The first step: tokenizing

```
tr -sc 'A-Za-z' '\n' < shakes.txt | head
```

```
THE
SONNETS
by
William
Shakespeare
From
fairest
creatures
We
...
```

The second step: sorting

```
tr -sc 'A-Za-z' '\n' < shakes.txt | sort | head
```

```
A
A
A
A
A
A
A
A
A
...
```

More counting

- Merging upper and lower case

```
tr 'A-Z' 'a-z' < shakes.txt | tr —sc 'A-Za-z' '\n' | sort | uniq —c
```

- Sorting the counts

```
tr 'A-Z' 'a-z' < shakes.txt | tr —sc 'A-Za-z' '\n' | sort | uniq —c | sort —n —r
```

```
23243 the
22225 i
18618 and
16339 to
15687 of
12780 a
12163 you
10839 my
10005 in
8954  d
```

What happened here?

Issues in Tokenization

- `Finland's capital` → `Finland Finlands Finland's ?`
- `what're, I'm, isn't` → `What are, I am, is not`
- `Hewlett-Packard` → `Hewlett Packard ?`
- `state-of-the-art` → `state of the art ?`
- `Lowercase` → `lower-case lowercase lower case ?`
- `San Francisco` → one token or two?
- m.p.h., PhD. → ??

Tokenization: language issues

- French
  - *L'ensemble* → one token or two?
    - *L* ? *L'* ? *Le* ?
    - Want *l'ensemble* to match with *un ensemble*
- German noun compounds not segmented
  - *Lebensversicherungsgesellschaftsangestellter*
  - 'life insurance company employee'
  - German information retrieval needs **compound splitter**

Tokenization: language issues

- Chinese and Japanese -- no spaces between words:
  - 莎拉波娃现在居住在美国东南部的佛罗里达。
  - 莎拉波娃 现在 居住 在 美国 东南部 的 佛罗里达
  - Sharapova now lives in US southeastern Florida
- Further complicated in Japanese, with multiple alphabets intermingled
  - Dates/amounts in multiple formats

フォーチュン500社は情報不足のため時間あた$500K(約6,000万円)

Katakana Hiragana Kanji Romaji

End-user can express query entirely in hiragana!

Word Tokenization in Chinese

- Also called **Word Segmentation**
- Chinese words are composed of characters
  - Characters are generally 1 syllable and 1 morpheme.
  - Average word is 2.4 characters long.
- Standard baseline segmentation algorithm:
  - Maximum Matching (also called Greedy)

Maximum Matching Word Segmentation Algorithm ("greedy")

Given a wordlist of Chinese, and a string.

1) Start a pointer at the beginning of the string
2) Find the longest word in dictionary that matches the string starting at pointer
3) Move the pointer over the word in string
4) Go to 2

Max-match segmentation illustration

- Thecatinthehat
- Thetabledownthere

the cat in the hat

the table down there

theta bled own there

Doesn't generally work in English!

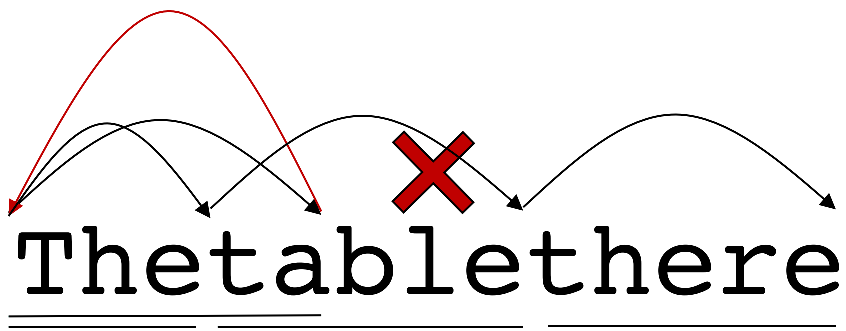
- But works astonishingly well in Chinese
  - 莎拉波娃现在居住在美国东南部的佛罗里达。
  - 莎拉波娃 现在 居住 在 美国 东南部 的 佛罗里达
- Modern probabilistic segmentation algorithms even better

Greedy matching

Thetabledownthere

Thetablethere

Backtracking



Dynamic programming

Keep track of intermediate results (segments of string that can be parsed as a sequence of words)

`Thetablethere`

Location (character indices)	Parse
0-3	the
0-5	theta
0-8	the + table
0-11	the + table + the
0-13	the + table + there

WORD NORMALIZATION & STEMMING

Normalization

- Need to "normalize" terms
  - Information Retrieval: indexed text & query terms must have same form.
    - We want to match *U.S.A.* and *USA*
- We implicitly define equivalence classes of terms
  - e.g., deleting periods in a term
- Alternative: asymmetric expansion:
  - Enter: *window* Search: *window, windows*
  - Enter: *windows* Search: *Windows, windows, window*
  - Enter: *Windows* Search: *Windows*
- Potentially more powerful, but less efficient

Case folding

- Applications like IR: reduce all letters to lower case
  - Since users tend to use lower case
  - Possible exception: upper case in mid-sentence?
    - e.g., *General Motors*
    - *Fed* vs. *fed*
    - *SAIL* vs. *sail*
- For sentiment analysis, MT, Information extraction
  - Case is helpful (*US* versus *us* is important)

Lemmatization

- Reduce inflections or variant forms to base form
  - *am, are, is* → *be*
  - *car, cars, car's, cars'* → *car*
- *the boy's cars are different colors* → *the boy car be different color*
- Lemmatization: have to find correct dictionary headword form
- Machine translation
  - Spanish quiero ('I want'), quieres ('you want') same lemma as querer 'want'

Morphology

- Morphemes:
  - The small meaningful units that make up words
  - Stems: The core meaning-bearing units
  - Affixes: Bits and pieces that adhere to stems
    - Often with grammatical functions

Stemming

- Reduce terms to their stems in information retrieval
- *Stemming* is crude chopping of affixes
  - language dependent
  - e.g., ***automate(s), automatic, automation*** all reduced to ***automat***.

for example compressed and compression are both accepted as equivalent to compress.

for exampl compress and compress ar both accept as equival to compress

Stemming not simple...

```
(*v*)ing → ø   walking   → walk
               sing      → sing
```

```
tr -sc 'A-Za-z' '\n' < shakes.txt | grep 'ing$' | sort | uniq -c | sort –nr
```

1312 King	548 being
548 being	541 nothing
541 nothing	152 something
388 king	145 coming
375 bring	130 morning
358 thing	122 having
307 ring	120 living
152 something	117 loving
145 coming	116 Being
130 morning	102 going

```
tr -sc 'A-Za-z' '\n' < shakes.txt | grep '[aeiou].*ing$' | sort | uniq -c | sort –nr
```

Complex morphology

- Some languages require complex morpheme segmentation
  - Turkish
  - Uygarlaştıramadıklarımızdanmışsınızcasına
  - `(behaving) as if you are among those whom we could not civilize'
  - Uygar `civilized' + laş `become'

    + tır `cause' + ama `not able'

    + dik `past' + lar 'plural'

    + ımız 'p1pl' + dan 'abl'

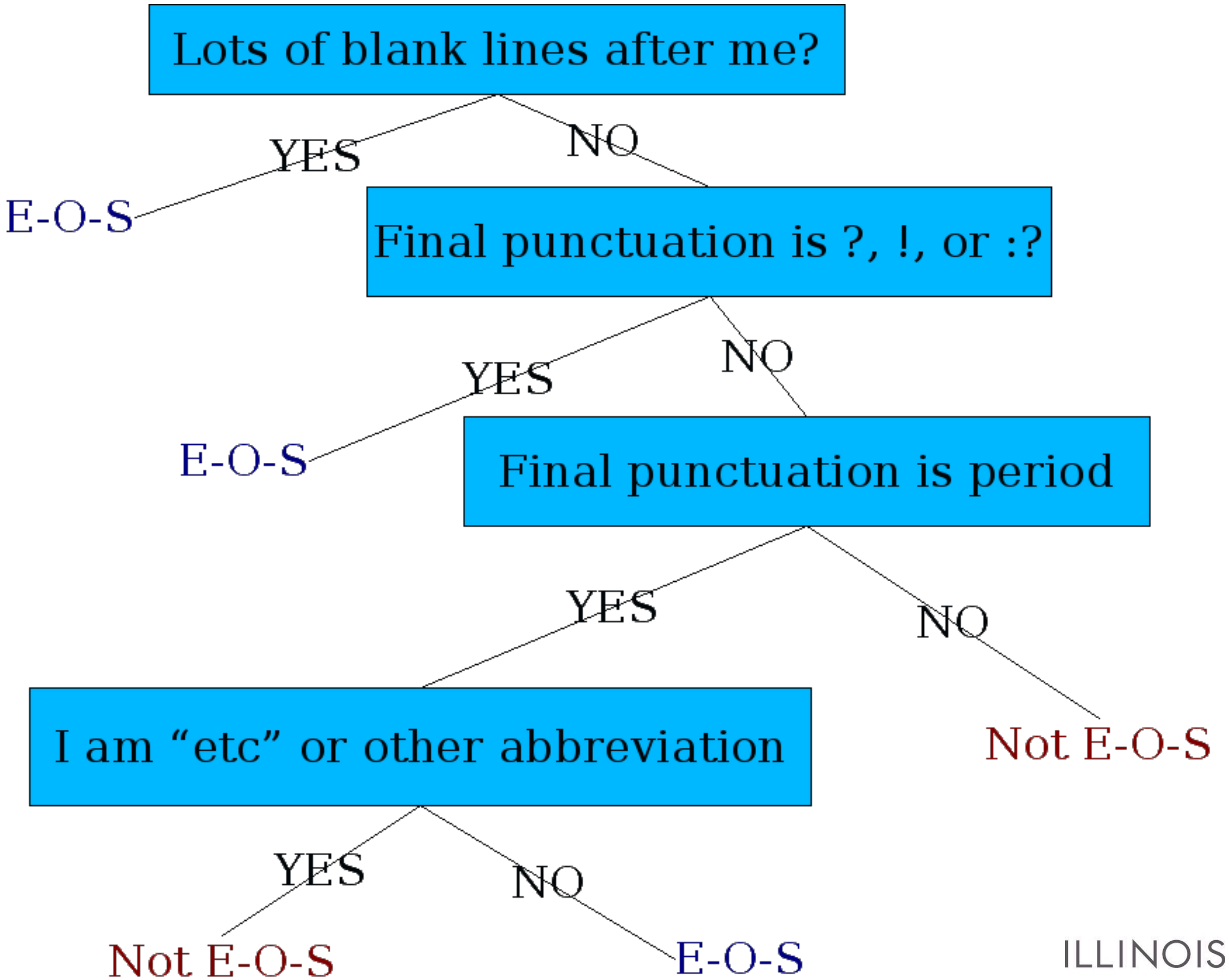
    + mış 'past' + sınız '2pl' + casına 'as if'

SENTENCE SEGMENTATION

Where to break sentences?

- !, ? are relatively unambiguous
- Period "." is <u>very</u> ambiguous
  - Sentence boundary
  - Abbreviations like Inc. or Dr.
  - Numbers like .02% or 4.3
- Build a classifier
  - Looks at a "."
  - Decides EndOfSentence/NotEndOfSentence
  - Classifiers: hand-written rules, regular expressions, or machine-learning

A Decision Tree



More sophisticated features

- Case of word preceding ".":
  Upper, Lower, Cap, Number
- Case of word following ".":
  Upper, Lower, Cap, Number

- Numeric features
  – Length of word preceding "."
  – Probability(word preceding "." occurs at end-of-sent)
  – Probability(word after "." occurs at beginning-of-s)

Implementing Decision Trees

- A decision tree is just an if-then-else statement
- The interesting question is choosing the features

- Setting up the structure is often too hard to do by hand
  - Only possible for very simple features, domains
    - For numeric features, it's too hard to pick each threshold
  - Instead, structure usually learned by machine learning from a training corpus (later in the course, and in CS 584)