

# Week 2: Basic Text Processing

Regular Expressions

# 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
<code>[wW]oodchuck</code>	Woodchuck, woodchuck
<code>[1234567890]</code>	Any digit

- Ranges `[A-Z]`

Pattern	Matches	
<code>[A-Z]</code>	An upper case letter	<u>D</u> renched Blossoms
<code>[a-z]</code>	A lower case letter	<u>m</u> y beans were impatient
<code>[0-9]</code>	A single digit	Chapter <u>1</u> : Down the Rabbit Hole

# Regular Expressions: Negation in Disjunction

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

Pattern	Matches	
<code>[^A-Z]</code>	Not an upper case letter	O <u>y</u> fn pripetchik
<code>[^Ss]</code>	Neither 'S' nor 's'	<u>I</u> have no exquisite reason"
<code>[^e^]</code>	Neither e nor ^	Look h <u>e</u> re
<code>a^b</code>	The pattern a carat b	Look up <u>a^b</u> now

# Regular Expressions: More Disjunction

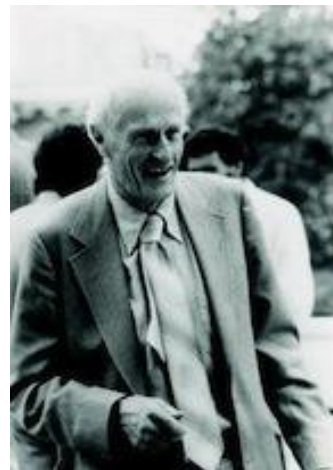
- Woodchucks is another name for groundhog!
- The pipe | for disjunction

Pattern	Matches
<code>groundhog woodchuck</code>	
<code>yours mine</code>	yours mine
<code>a b c</code>	= <code>[abc]</code>
<code>[gG]roundhog [Ww]oodchuck</code>	



# Regular Expressions: ? \* + .

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



Stephen C Kleene

Kleene \*, Kleene +

# Regular Expressions: Anchors ^ \$

Pattern	Matches
<span style="color: red;">^</span> [A-Z]	<u>P</u> alo Alto
<span style="color: red;">^</span> [^A-Za-z]	<u>1</u> <u>"Hello"</u>
\. <span style="color: red;">\$</span>	The end <u>.</u>
. <span style="color: red;">\$</span>	The end <u>?</u> The end <u>!</u>

# Example

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

the

Misses capitalized examples

[tT]he

Incorrectly returns other or theology

[^a-zA-Z] [tT]he [^a-zA-Z]



# 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 precision (minimizing false positives)
  - Increasing coverage or recall (minimizing false negatives).

# Summary

- Regular expressions play a surprisingly large role
  - Sophisticated sequences of regular expressions are often the first model for any text processing text
- For many hard tasks, we use machine learning classifiers
  - But regular expressions are used as features in the classifiers
  - Can be very useful in capturing generalizations

# Basic Text Processing

Word 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, 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 (or 14)
  - 13 types (or 12) (or 11?)

# How many words?

***N*** = number of tokens

***V*** = vocabulary = set of types

$|V|$  is the size of the vocabulary

Church and Gale (1990):  $|V| > O(N^{1/2})$

	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
```

Change all non-alpha to newlines

```
    | sort
```

Sort in alphabetical order

```
    | uniq -c
```

Merge and count each type

1945 A	25 Aaron
72 AARON	6 Abate
19 ABBESS	1 Abates
5 ABBOT	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

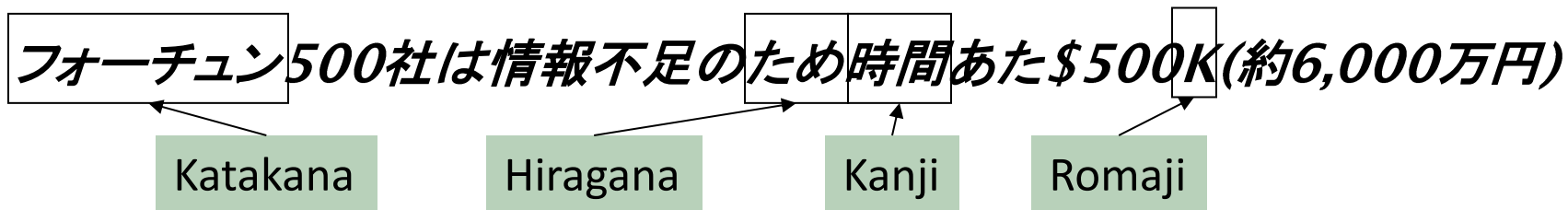
- Finland's capital → Finland Finlands Finland's ?
- what're, I'm, isn't → What are, I am, is not
- Hewlett-Packard → Hewlett Packard ?
- state-of-the-art → state of the art ?
- Lowercase → lower-case lowercase lower case ?
- San Francisco → **one token or two?**
- m.p.h., PhD. → ??

# Tokenization: language issues

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

# Tokenization: language issues

- Chinese and Japanese no spaces between words:
  - 莎拉波娃現在居住在美國東南部的佛羅里達。
  - 莎拉波娃 現在 居住 在 美國 東南部 的 佛羅里達
  - Sharapova now lives in US southeastern Florida
- Further complicated in Japanese, with multiple alphabets intermingled
  - Dates/amounts in multiple formats



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

- 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                      the cat in the hat
- Thetabledownthere                the table down there  
    theta bled own there
- Doesn't generally work in English!
- But works astonishingly well in Chinese
  - 莎拉波娃現在居住在美國東南部的佛羅里達。
  - 莎拉波娃 現在 居住 在 美國 東南部 的 佛羅里達
- Modern probabilistic segmentation algorithms even better

# Basic Text Processing

Word Normalization and  
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 equal to compress



# Porter's algorithm

## The most common English stemmer

### Step 1a

sses	→ ss	caresses	→ caress
ies	→ i	ponies	→ poni
ss	→ ss	caress	→ caress
s	→ ∅	cats	→ cat

### Step 2 (for long stems)

ational	→ ate	relational	→ relate
izer	→ ize	digitizer	→ digitize
ator	→ ate	operator	→ operate
...			

### Step 1b

(*v*)ing	→ ∅	walking	→ walk
		sing	→ sing
(*v*)ed	→ ∅	plastered	→ plaster
...			

### Step 3 (for longer stems)

al	→ ∅	revival	→ reviv
able	→ ∅	adjustable	→ adjust
ate	→ ∅	activate	→ activ
...			



# Viewing morphology in a corpus

## Why only strip –ing if there is a vowel?

( $*v^*$ ) ing  $\rightarrow \emptyset$    walking  $\rightarrow$  walk  
                        sing  $\rightarrow$  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
```

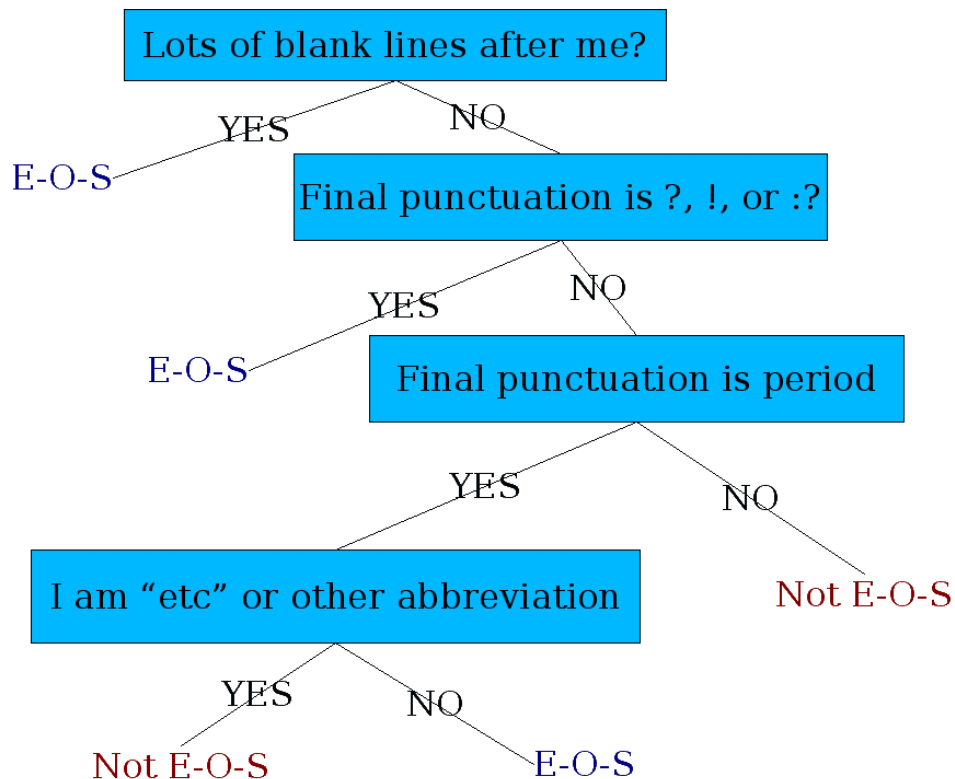
# Basic Text Processing

Sentence Segmentation  
and Decision Trees

# Sentence Segmentation

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

# Determining if a word is end-of-sentence: a Decision Tree



# More sophisticated decision tree features

- Case of word with “.”: Upper, Lower, Cap, Number
- Case of word after “.”: Upper, Lower, Cap, Number
- Numeric features
  - Length of word with “.”
  - Probability(word with “.” occurs at end-of-s)
  - Probability(word after “.” occurs at beginning-of-s)

# Implementing Decision Trees

- A decision tree is just an if-then-else statement
- The interesting research is choosing the features
- Setting up the structure is often too hard to do by hand
  - Hand-building 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



# Decision Trees and other classifiers

- We can think of the questions in a decision tree
- As features that could be exploited by any kind of classifier
  - Logistic regression
  - SVM
  - Neural Nets
  - etc.