

# CMPN463: Natural Language Processing



## Lecture 01: Basic Text Processing

Mohamed Alaa El-Dien Aly  
Computer Engineering Department  
Cairo University  
Fall 2013

# Agenda

- Regular Expressions
- Word Tokenization
- Word Normalization
- Sentence Segmentation

## **Acknowledgment:**

Most slides adapted from Chris Manning and Dan Jurafsky's NLP class on [Coursera](#).






# Regular expressions

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



# RegexPal

← → ↻ regexpal.com

 **regexpal** 0.1.4 – a JavaScript regular expression tester

Case insensitive (i)  ^\$ match at line breaks (m)  Dot matches all (s; via [XRegExp](#))

We looked!  
And we saw him step in on the mat!  
We looked!  
And we saw him!  
The Cat in the Hat!



# 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	
[ ^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



# 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>	



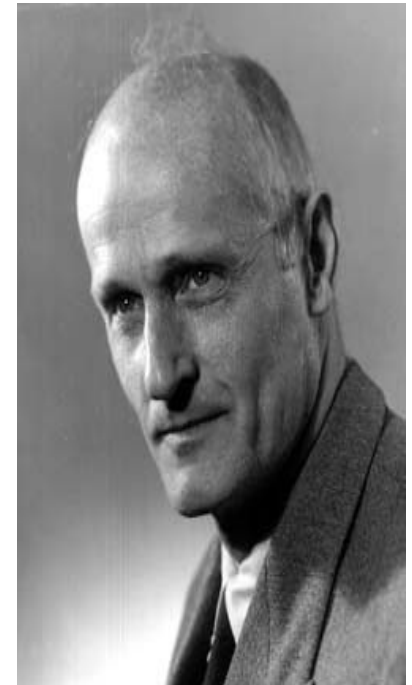
Photo D. Fletcher





# 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>	any character	<u>begin</u> <u>begun</u> <u>begun</u> <u>beg3n</u>



Stephen C Kleene

Kleene \*, Kleene +



# Regular Expressions: Anchors <sup>^</sup> <sup>\$</sup>

<sup>^</sup>: beginning of the line

<sup>\$</sup>: end of the line

Pattern	Matches
<sup>^</sup> [A-Z]	<u>P</u> alo Alto
<sup>^</sup> [ <sup>^</sup> A-Za-z]	<u>1</u> <u>"Hello"</u>
\. <sup>\$</sup>	The end <u>.</u>
<sup>.</sup> <sup>\$</sup>	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)**

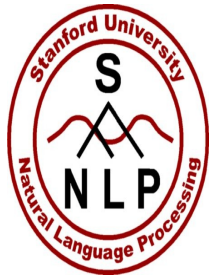


## Errors cont.

- 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).

# Practical Regular Expression

- Used in Lexical Analysis for e.g.
  - Parsers
  - Compilers
- More in the **Compilers** class
  - Lex/Flex program



# 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







## Text Normalization

- Every NLP task needs to do text normalization:
  - Segmenting/tokenizing words in running text
  - Normalizing word formats
  - 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)                      “San Francisco” vs “San” “Francisco”
  - 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
```

```
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  
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. → ??



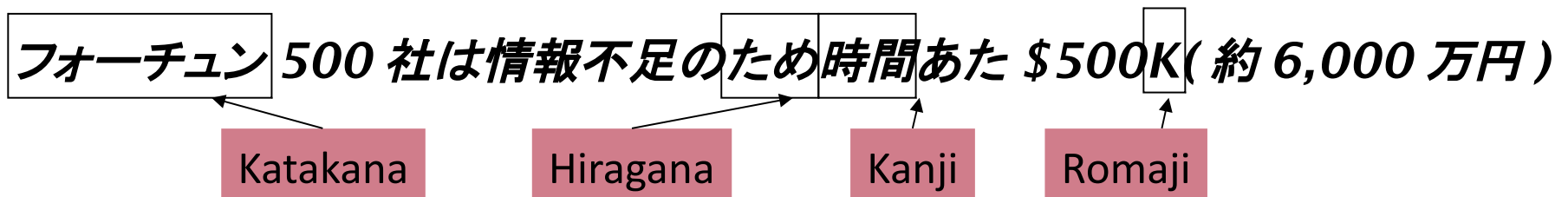
# Tokenization: language issues

- 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 (cont.)

- 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



End-user can express query entirely in hiragana!





# 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



## 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 (simplified *lemmatization*)
  - 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 example compress and compress ar both accept as equal to compress







# Viewing morphology in a corpus

## Why only strip `-ing` if there is a vowel?

```
(*v*)ing → ∅ 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
```



# Dealing with complex morphology is sometimes necessary

- Some languages requires complex morpheme segmentation
  - Turkish
  - **Uygarlastiramadiklarimizdanmissinizcasina**
  - `(behaving) as if you are among those whom we could not civilize`
  - **Uygar** `civilized` + **las** `become`
    - + **tir** `cause` + **ama** `not able`
    - + **dik** `past` + **lar** `plural`
    - + **imiz** `p1pl` + **dan** `abl`
    - + **mis** `past` + **siniz** `2pl` + **casina** `as if`



# Basic Text Processing

Sentence Segmentation  
and Decision Trees

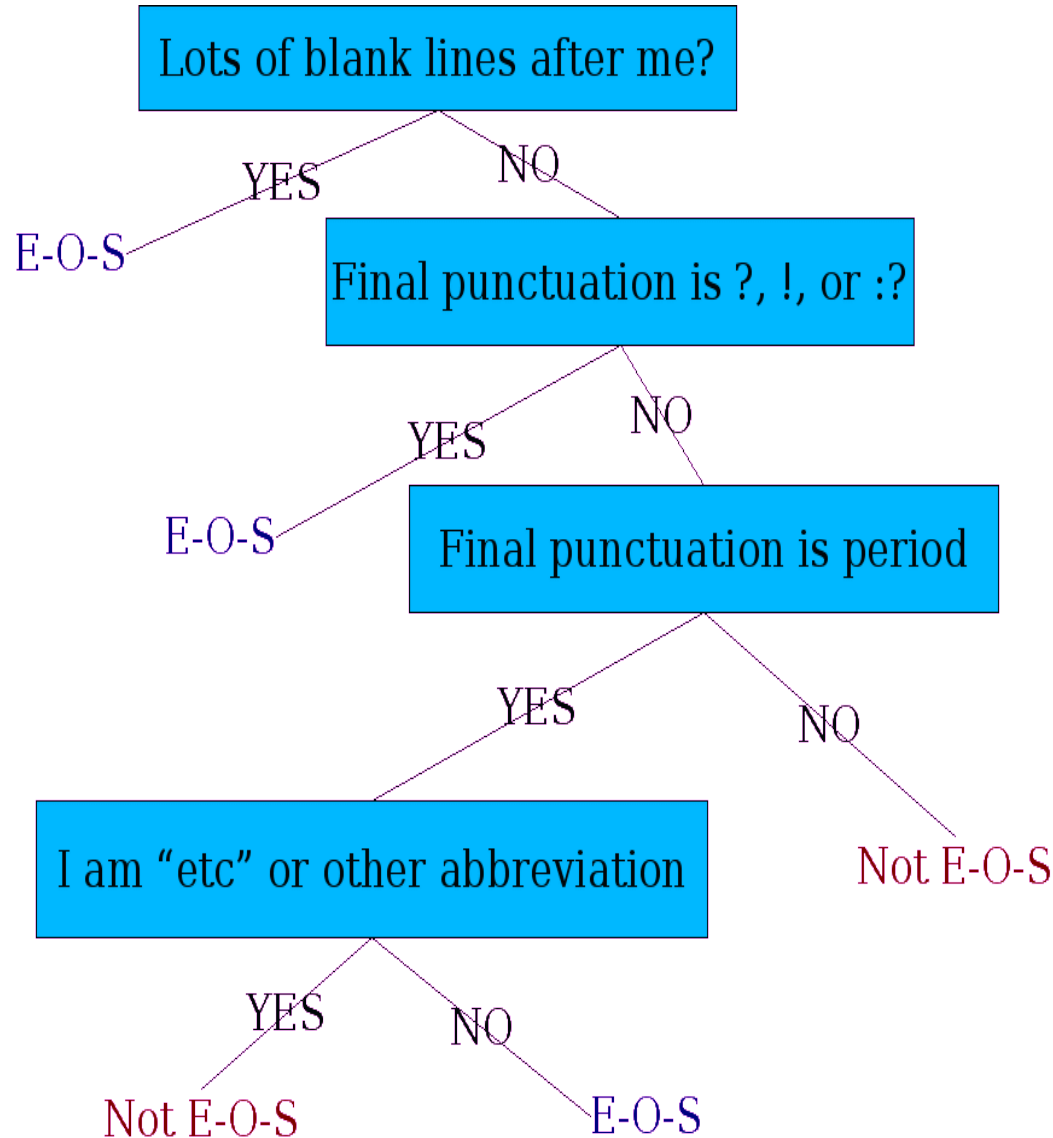


# Sentence Segmentation

- !, ? 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.

# Recap

- Regular Expressions
- Word Tokenization
- Word Normalization
- Sentence Segmentation