Comp 408
Advanced
Topics in
Artificial
Intelligence

Lecture 2

Regular Expression 2 & Text Processing

15 / 2 / 2025

Some Slides are by D. Jurafsky and J. M. Martin

#### Agenda

- Regular Expression
- Python Implementation
- Words and Corpora
- Text Preprocessing
  - 1. Word Tokenizing
  - 2. Byte Pair Encoding (BPE)
  - 3. Normalizing word formats
  - 4. Lemmatization and stemming
  - 5. Segmenting sentences.

#### Summary of RE

- Parenthesis
- Counters
- Sequences and anchors
- Disjunction

```
()
* + ? { }
the ^myend$
|, []
```

• Period (.) Marches any single character.

#### More operators

| RE                     | Match  |
|------------------------|--|
| \*                     | an asterisk "*"                              |
| \.                     | a period "."                                 |
| \?                     | A question mark                              |
| \d                     | [0-9], any digit                             |
| \D                     | [^0-9], any non-digit                        |
| $\setminus \mathbf{w}$ | [a-zA-Z0-9_], any alphanumeric or underscore |
| $\setminus \mathbf{W}$ | [^\w], a non-alphanumeric e.g., ?; +* -      |
| \s                     | $[\r\h], white space$                        |
| \S                     | [^\s], non-whitespace                        |

#### Example

Write regular expression to matches all computers with the following features:

- at least 6 GHz and
- 500 GB of disk space
- for less than \$1000

#### Example (cont.): Give RE to represent numbers less than \$1000

- To represent numbers beginning with \$:
  - **-** /\\$[0-9]+/
- For 2 digits decimal number:
  - **-** /\\$[0-9]+\.[0-9][0-9]/
- The decimal number can be optional:
  - $/ \$[0-9] + ( \cdot .[0-9][0-9])? /$
- The numbers should not be proceeded by letters
  - $-/(^{\ }\ ^{\ }\ ^{\ })$
- Numbers should be less than 1000:
  - $-/(^{\mathbb{W}})\$ [0-9]{1, 3}(\.[0-9][0-9])?\b/

#### Example (cont.): Give RE to represent at Least 6 GHz

```
• /([6-9]|[1-9][0-9]+)(\.[0-9]+)? *(GHz|[Gg]igahertz)/
```

6 to 9

10 to 99

```
/([6-9]|[0-9]\d) (\.[0-9]+)? *(GHz|[Gg]igahertz)/g
Test String
5 GHz
```

#### Example (cont.): Give RE to represent from 500-999 GB

• /[5-9][0-9][0-9] \*(GB|[Gg]igabytes?)/

## Python Implementation

#### Searching for a given string in python

To search for the string phone in the text 'My phone number is ...'

```
'phone' in 'My phone number is 012-233-23454'
```

True

To search for a given phone number in the text

```
'012-233-23454' in 'My phone number is 012-233-23454'
```

True

#### Searching for a given pattern in python

- If we want to search for any telephone number, we can use regular expression to represent telephone numbers pattern.
- First import the regular expression package (re)

```
import re
  text1 = 'My phone number is 012-233-23454, another number 011 222 22222'
                                                             R or r is used before the regular expression
   pattern = r'\d{3}[ -/]?\d{3}[ -/]?\d{5}'
                                                             pattern to indicate that it is a raw string
    re.search(pattern, text1)
    <re.Match object; span=(19, 32), match='012-233-23454'>
The telephone number spans from character 19 to 32 in the text
```

#### Note

• Function search displays only the first match:

| search(pattern, string) | Scan through a string to find the first match only of the pattern |
|-------------------------|---|
|                         |   |

#### Function findall displays all matches

```
import re

text1 = 'My phone number is 012 233 23454, another number 01122222222'

pattern = r'\d{3}[ -/]?\d{3}[ -/]?\d{5}'

re.findall(pattern, text1)

['012 233 23454', '01122222222']
```

findall(pattern, string)

Find all matched strings and returns the result as list.

## Write regular expression to match a valid date of the form 10-02-2024, 10/02/2024, 10/02/24, or 02/10 2024

```
text1 = "In 10-02-2024 the spring term of the acedimic year 2023-2024 starts. \n Saturday 10/02/2

print(text1)

In 10-02-2024 the spring term of the acedimic year 2023-2024 starts.
    Saturday 10/02/24 at 8.00 AM is our first COMP408 Lecture. In 10 2 2024 is our second lecture

import re

re.findall(r'\d{1,2}[\s/-]\d{1,2}[\s/-]\d{2,4}', text1)

['10-02-2024', '10/02/24', '10 2 2024']
```

# Find regular expression to match the strings {heat, cheat, meat, seat, great, eat}

RE:  $\sqrt{b(m|s|c?h|gr)?eatb/}$ 

In python:

```
| reg1 = r'\b(m|s|c?h|gr)?eat\b'
| tex2 ='... meat, ... great, ... eat, heat, cheat, seat'
| re.findall(reg1, tex2)
|: ['m', 'gr', '', 'h', 'ch', 's']
```

The output contains only the strings inside the parenthesis (, )

#### Important Note

- Parentheses have double functions in regular expressions:
  - 1. Used to group items to specify the order in which operators should apply.
  - 2. Used to capture something is registers.
- If we want to use parenthesis for grouping, we should use the non-capturing group by putting ?: after the parenthesis (?: pattern)
- In the last Pythone example: reg1 = r'\b(?:m|s|c?h|gr)?eat\b'

```
reg1 = r'\b(?:m|s|c?h|gr)?eat\b'

tex2 ='... meat, ... great, ... eat, heat, cheat, seat'

re.findall(reg1, tex2)

['meat', 'great', 'eat', 'heat', 'cheat', 'seat']
```

# Write a regular expression to match a valid date of the form 10 Feb 2024 or 10 February 2024 in a given text

```
import re
▶ text1 = "In 08 Feb 25 the spring term of the academic year 2024-2025 starts. \n Saturday 08 February
 print(text1)
  In 08 Feb 25 the spring term of the academic year 2024-2025 starts.
   Saturday 08 February 25 at 8.00 AM is our first COMP408 Lecture. In 15 Feb 24 is our second lecture

    | re.findall(r'\d{2} (Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec)[a-z]* \d{2,4}', text1)
  ['Feb', 'Feb', 'Feb']
```

Only the string in the parentheses appears, to fix this error add ?: at the beginning of the "(" parentheses

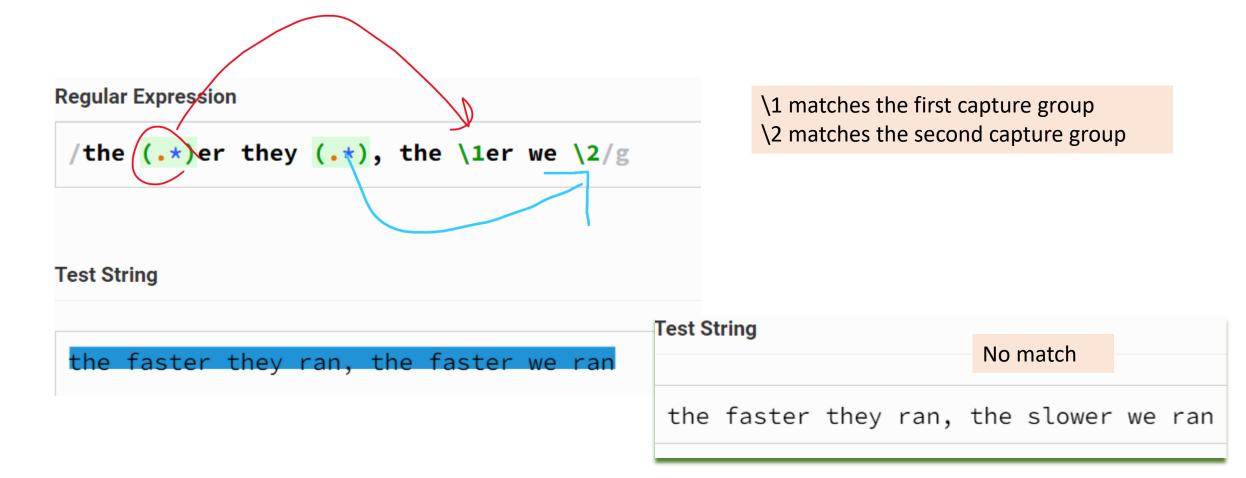
## Write a regular expression to match a valid date of the form 10 Feb 2024 or 10 February 2024 in a given text

Add the non-capturing group: ?:

```
re.findall(r'\d{2} (?:Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec)[a-z]* \d{2,4}', text1)
['08 Feb 25', '08 February 25', '15 Feb 24']
```

#### Using RE to capture strings in register

Capturing group is the use of parentheses to store a pattern in memory



#### Example 1

- Use register to put angle brackets around all integers in a text:
  - for example, change 5 books to <5> books

```
st = 'I bought 5 computer since books, 1 for Artificial intelligence 2 for machine learning'

pa1 = r'(\d+)'

pa2 = r'<\1>'

re.sub(pa1, pa2, st)
```

'I bought <5> computer since books, <1> for Artificial intelligence <2> for machine learning'

#### Example 2



#### Question

What are the strings matched by each of following regular expressions:

- 1.  $\sqrt{w+-\sqrt{w+}}$
- 2.  $\w+\@\w+\.\w{3}$

\w: [a-zA-Z0-9\_]

#### Substitutions

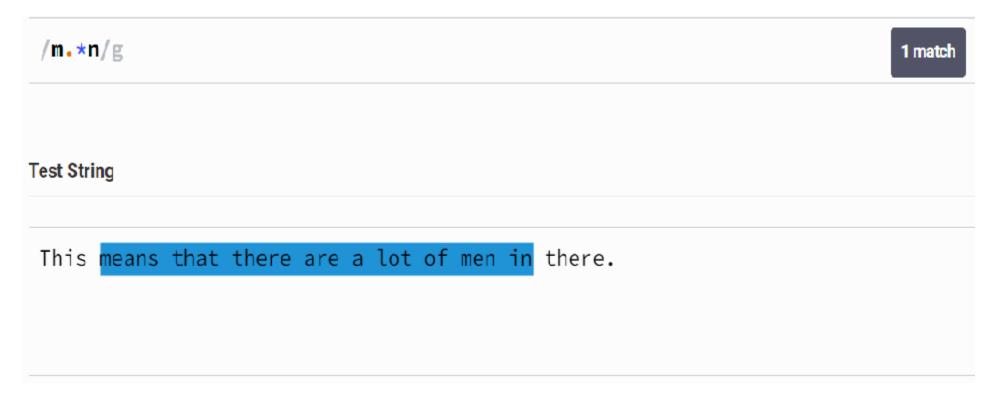
- An important use of regular expression is in substitution
- Python code for substitute the words colour or colours by color

```
import re
pattern = "colours?"
rep = "color"
Str = 'red colours, black colour'
newS = re.sub(pattern, rep, Str)
print(newS)
```

Output: red color, black color

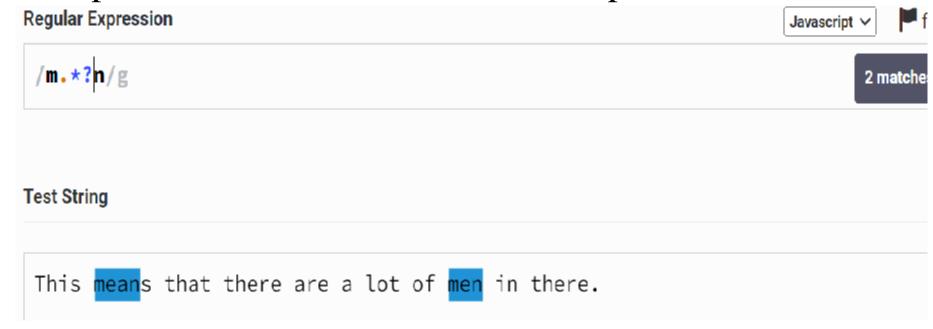
#### Greedy regular expression match

• Regular expressions always match largest string they can; that pattern is called greedy, expanding to cover as much of a string as they can.



#### Non-greedy (reluctant) match

- To enforce non-greedy matching, use \*? Instead of \* which matches as little text as possible.
- The operator +? matches as little text as possible.



#### Greedy and non-greedy RE

| Greedy | Non-greedy |                         |
|--------|------------|-------------------------|
| *      | *?         | Zero or more characters |
| +      | +?         | One or more characters  |
| {n, m} | {n, m}?    | From n to m characters  |
| ?      | ??         | 0 or 1 characters       |

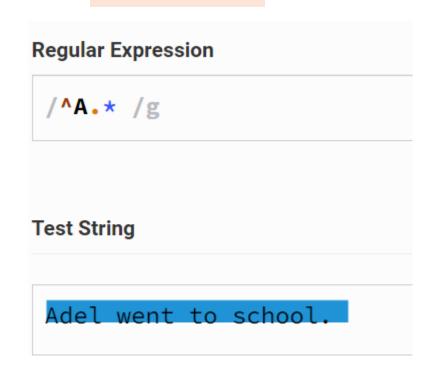
Example 1: search for a string begins with A then any characters then space.

Regular Expression

/^A.\*? /g

Test String

Adel\_went to school.



Greedy

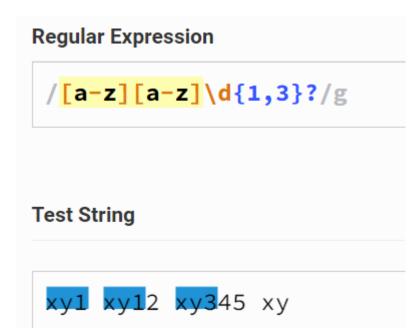
#### Example 1 (Cont.):

```
| import re
text = 'Adel went to school. '
  b = re.search('^A.*?', text)
                                          Non greedy match;
                                          until the first space
Н
  b
  <re.Match object; span=(0, 5), match='Adel '>
                                         Greedy match; until
  b = re.search('^A.*', text)
                                         the last space
  b
  <re.Match object; span=(0, 21), match='Adel went to school. '>
```

# Example 2: find two characters followed by 1,2, or 3 digits

#### Example 2 Cont.

# Regular Expression /[a-z][a-z]\d{1,3}/g Test String xy1 xy12 xy345 xy



# Example 3: find two characters followed by 0 or 1 digit

```
text = 'ab1 xy34 cd56 yy6'

re.findall("[a-z][a-z]\d?", text) greedy

['ab1', 'xy3', 'cd5', 'yy6']

re.findall("[a-z][a-z]\d??", text) Non greedy

['ab', 'xy', 'cd', 'yy']
```

### Words and Corpora

#### How many words?

- I do uh main- mainly business data processing
  - Fragments (e.g., main), filled pauses (e.g., uh)
- Seuss's cat in the hat is different from other cats!
  - Lemma: the base form of a word, cat and cats have the same lemma cat:
    - the same stem,
    - part of speech (POS)- both are noun,
    - same word sense
  - Wordform: the full inflected or derived form of the word
    - cat and cats = different wordforms

#### Corpus (Plural corpora )

- A computer-readable collection of text or speech.
- The Brown corpus is a million-word collection of samples from 500 written texts from different genres (newspaper, fiction, academic, etc.), assembled at Brawn University in 1963-1964.

#### How many words?

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

- To count the number of words in an English statement we need to know the difference between types and tokens:
- **Type**: an element of the vocabulary.
  - How many distinct words in a corpus?
- Token: an instance of that type in running text.
- How many? (It depends on your counts)
  - 15 tokens (or 14 if San Francisco is one word)
  - 13 types (or 12 if San Francisco is one word)

(or 11? If they and their are considered the same lemma)

#### How many words in a corpus?

N = number of tokens

V = vocabulary = set of types, |V| is size of vocabulary

Heaps Law (Herdan's Law),  $|V| = kN^{\beta}$  where often .67 <  $\beta$  < .75

i.e., vocabulary size grows with > square root of the number of word tokens

| Corpus                          | Tokens = N  | Types = $ V $ |
|---------------------------------|-------------|---------------|
| Switchboard phone conversations | 2.4 million | 20 thousand   |
| Shakespeare                     | 884,000     | 31 thousand   |
| COCA                            | 440 million | 2 million     |
| Google N-grams                  | 1 trillion  | 13+ million   |

# Text Preprocessing

# Space-based tokenization

- A very simple way to tokenize
  - For languages that use space characters between words
    - Arabic, Cyrillic, Greek, Latin, etc., based writing systems
  - Segment off a token between instances of spaces
- White space is not always sufficient
  - Words like Adel's is considered as one word.
  - I'm need to be separated as I and am.
  - New York is considered as two words.

### Issues in Tokenization

- Can't just blindly remove punctuation:
  - m.p.h., Ph.D., AT&T, cap'n
  - prices (\$45.55)
  - dates (01/02/06)
  - URLs (http://www.stanford.edu)
  - hashtags (#nlproc)
  - email addresses (someone@cs.colorado.edu)
- Clitic: a word that doesn't stand on its own
  - "are" in we're,
  - "am" in I'm. French "je" in j'ai, "le" in l'honneur
- When should multiword expressions (MWE) be words?
  - New York, rock 'n' roll

# Using natural language toolkit (nltk) python library to tokenize text

```
Import nltk

Itext = 'that U.S.A. poster-print costs $12.40 ...'

Import nltk

pattern = r'(?:[A-Z]\.)+|\w+(?:-\w+)*| \$?\d+(?:\.\d+)?%?| \.\.\.'

Inport nltk.regexp_tokenize(text, pattern)

['that', 'U.S.A.', 'poster-print', 'costs', '$12.40', '...']
```

### Notes

- A tokenizer can also be used to **expand clitic** contractions that are marked by apostrophes:
  - Converting we're to we are
- Ambiguity that arises from using apostrophe, ', should be handled:
  - Genitive marker: Adam's book
  - Quotative: 'The other class', she said
  - Clitic: they're

# Tokenization in languages without spaces

Many languages (like Chinese, Japanese, Thai) don't use spaces to separate words!

How do we decide where the token boundaries should be?

### Word tokenization in Chinese

Chinese words are composed of characters called "hanzi" (or sometimes just "zi")

Each one represents a meaning unit called a morpheme.

Each word has on average 2.4 of them.

But deciding what counts as a word is complex and not agreed upon.

•姚明进入总决赛 "Yao Ming reaches the finals"

- •姚明进入总决赛 "Yao Ming reaches the finals"
- •3 words?
- •姚明 进入 总决赛
- YaoMing reaches finals
- •5 words?

- •姚明进入总决赛 "Yao Ming reaches the finals"
- •3 words?
- •姚明 进入 总决赛
- YaoMing reaches finals
- •5 words?
- •姚 明 进入 总 决赛
- Yao Ming reaches overall finals
- •7 characters? (don't use words at all):

- •姚明进入总决赛 "Yao Ming reaches the finals"
- •3 words?
- •姚明 进入 总决赛
- YaoMing reaches finals
- •5 words?
- •姚 明 进入 总 决赛
- Yao Ming reaches overall finals
- •7 characters? (don't use words at all):
- •姚 明 进 入 总 决 赛
- •Yao Ming enter enter overall decision game

# Word tokenization / segmentation

So in Chinese it's common to just treat each character (zi) as a token.

• So the **segmentation** step is very simple

In other languages (like Thai and Japanese), more complex word segmentation is required.

• The standard algorithms are neural sequence models trained by supervised machine learning.

# Basic Text Processing

Byte Pair Encoding (BPE)

# Another option for text tokenization

Instead of

- white-space segmentation
- single-character segmentation (as in Chinese)

Use the **data** to tell us how to tokenize.

**Subword tokenization** (because tokens can be parts of words as well as whole words- the token new can be part of the word newer as well as the word new)

### Subword tokenization

- Three common algorithms:
  - 1. Byte-Pair Encoding (BPE) (Sennrich et al., 2016)
  - 2. Unigram language modeling tokenization (Kudo, 2018)
  - 3. WordPiece (Schuster and Nakajima, 2012)
- All the three algorithms have 2 parts:
  - A token learner that takes a raw training corpus and induces a vocabulary (a set of tokens).
  - A token **segmenter** that takes a raw test sentence and tokenizes it according to that vocabulary

# Byte Pair Encoding (BPE) token learner

Let vocabulary be the set of all individual characters

$$= \{A, B, C, D, ..., a, b, c, d....\}$$

- Repeat:
  - Choose the two symbols that are most frequently adjacent in the training corpus (say 'A', 'B')
  - Add a new merged symbol 'AB' to the vocabulary
  - Replace every adjacent 'A' 'B' in the corpus with 'AB'.
- Until *k* merges have been done.

# BPE token learner algorithm

```
function BYTE-PAIR ENCODING(strings C, number of merges k) returns vocab V
```

```
V \leftarrow all unique characters in C # initial set of tokens is characters

for i = 1 to k do # merge tokens til k times

t_L, t_R \leftarrow Most frequent pair of adjacent tokens in C

t_{NEW} \leftarrow t_L + t_R # make new token by concatenating

V \leftarrow V + t_{NEW} # update the vocabulary

Replace each occurrence of t_L, t_R in C with t_{NEW} # and update the corpus

return V
```

# Byte Pair Encoding (BPE) Addendum

Most subword algorithms are run inside space-separated tokens.

So, we commonly first add a special end-of-word symbol '\_\_\_' before space in training corpus

Next, separate into letters.

### BPE token learner

Original (very simple) corpus:

low low low low lowest lowest newer newer newer newer newer newer wider wider new new

Add end-of-word tokens, resulting in this vocabulary:

```
vocabulary
```

\_, d, e, i, l, n, o, r, s, t, w

### BPE token learner

Original (very simple) corpus:

low low low low lowest lowest newer newer newer newer newer newer wider wider new new

Add end-of-word tokens, resulting in this vocabulary:

#### 

### BPE token learner

#### corpus

#### 

```
6 newer_
```

- 3 wider\_
- $2 \quad \mathsf{new} \perp$

#### Merge e r to er

#### corpus

#### vocabulary

```
_, d, e, i, l, n, o, r, s, t, w
```

The most frequent pairs of adjacent symbols is er because it occurs in newer (frequency of 6) and wider (frequency of 3), total is 9 occurrences

#### vocabulary

```
_, d, e, i, l, n, o, r, s, t, w, er
```

Add the merged characters er to vocabulary and modify corpus

### **BPE**

#### corpus

Merge er \_ to er\_

#### corpus

#### vocabulary

\_, d, e, i, l, n, o, r, s, t, w, er

The most frequent pairs of adjacent symbols is ertotal is 9 occurrences

#### vocabulary

\_, d, e, i, l, n, o, r, s, t, w, er, er\_

### **BPE**

```
      corpus
      vocabulary

      5
      1 o w __
      __, d, e, i, 1, n, o, r, s, t, w, er, er__

      2
      1 o w e s t __

      6
      n e w er__

      3
      w i d er__

      2
      n e w __
```

Merge n e (total count of 8) to ne

### **BPE**

The next merges are:

# BPE token segmenter algorithm

- 1. Segment each test sentence into characters.
- 2. Then apply the rules made in training corpus to test corpus in order:
  - 1. Replace e r with er
  - 2. Replace er \_ with er\_
  - 3. Replace n e with ne
  - 4. Replace ne w with new
  - 5. Replace 1 o with lo
  - 6. Replace lo w with low
  - 7. Replace new er\_ with newer\_
  - 8. Replace low with low\_
  - 9. ...

# BPE token segmenter algorithm

On the test data, run each merge learned from the training data:

- Greedily
- In the order we learned them
- (test frequencies don't play a role)

So: merge every e r to er, then merge er \_ to er\_, etc.

- Result:
  - Test set "n e w e r \_" would be tokenized as a full word
  - Test set "lower\_" would be two tokens: "lower\_", since "lower\_" is unknow word (not among the vocabulary)

## Properties of BPE tokens

Usually include frequent words

And frequent subwords

• Which are often morphemes like *-est* or *-er* 

A morpheme is the smallest meaning-bearing unit of a language

• *unlikeliest* has 3 morphemes *un-*, *likely*, and *-est* 

### Normalization

- Need to "normalize" terms
  - Information Retrieval (IR): 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 (deleting dots in U.S.A. to match USA)
- Alternative: asymmetric expansion:
  - Enter: window Search: window, windows
  - Enter: windows Search: Windows, windows, window
  - Enter: *Windows* Search: *Windows*

# Case folding

- Applications like Information Retrieval: reduce all letters to lower case
  - Since users tend to use lower case
  - Possible exception: upper case in mid-sentence?
    - e.g., General Motors (the beginning of each word is capital)
    - Fed vs. fed (federal reserve system vs. the past of feed)
    - SAIL vs. sail (Stanford Artificial Intelligence laboratory vs the verb sail)
- For sentiment analysis, machine translation-MT, Information extraction
  - Case is helpful (*US* versus *us* is important)

### Lemmatization

Represent all words as their lemma, their shared root

- = dictionary headword form:
- am, are, is  $\rightarrow be$
- car, cars, car's, cars'  $\rightarrow$  car
- the boy's cars are different colors  $\rightarrow$  the boy car be different color
- Lemmatization: have to find correct dictionary headword form

# Lemmatization is done by Morphological Parsing

- Morphemes:
  - The small meaningful units that make up words
  - Stems: The core meaning-bearing units
  - Affixes: Parts that adhere to stems, often with grammatical functions
- Morphological Parsers (takes a word and parses it into morphemes)
  - Parse *cats* into two morphemes *cat* and *s*
  - Parse نهب into two morphemes في and في and

# Stemming

• Reduce terms to stems, chopping off affixes crudely

This was not the map we found in Billy Bones's chest, but an accurate copy, complete in all things-names and heights and soundings-with the single exception of the red crosses and the written notes.



Thi wa not the map we found in Billi Bone s chest but an accur copi complet in all thing name and height and sound with the singl except of the red cross and the written note

•

### Porter Stemmer

- Based on a series of rewrite rules run in series
  - A cascade, in which output of each pass fed to next pass
- Some sample rules:

```
ATIONAL \rightarrow ATE (e.g., relational \rightarrow relate)

ING \rightarrow \epsilon if stem contains vowel (e.g., motoring \rightarrow motor)

SSES \rightarrow SS (e.g., grasses \rightarrow grass)
```

# Sentence Segmentation

- !, ? mostly unambiguous but **period** "." is very ambiguous
  - Sentence boundary
  - Abbreviations like Inc. or Dr.
  - Numbers like .02% or 4.3

Common algorithm: Tokenize first: use rules or Machine Learning (ML) to classify a period as either (a) part of the word or (b) a sentence-boundary.

An abbreviation dictionary can help

Sentence segmentation can then often be done by rules based on this tokenization.

### Determining if a word is end-ofsentence: a Decision Tree

