

# **Lecture 2: Basic Text Processing**

## Tokenization and Normalization of Text

Some content in these slides was adapted from J&M 3rd ed. and from Wei Xu's course.

# **Text comes in many shapes ...**

The word "genius" is one of the most misused terms in history.

While it's often referenced accurately, the connotation that we commonly associate with it diverges away from the truth.

We correctly label intellectual brilliance and creative power as genius—and we should—but it's about time we stopped assuming that those things arise from talent or inborn giftedness alone.

# Or for example ...

1) Who was the last dialogue with? ✨ 2) Who is your first friend? 👤 3) Do you have many friends? 🧑 4) Favorite song? 🎵 5) Favorite movie? 🎬 6) Currently in love? ❤️ 7) Favorite season of the year? 🌳 8) Do you dance? 💃 9) Favorite fruit? 🍎 Share this to everyone you are following

3 days ago

1) with daisy 2) max 4) yes 5) basta - samsara 6) pile 7) yes 8) summer 9) yes 10) apple. I will not share, I'm a rebel! 😊😊😊



...

# Or for example ...



TroyTube YT 43 minutes ago

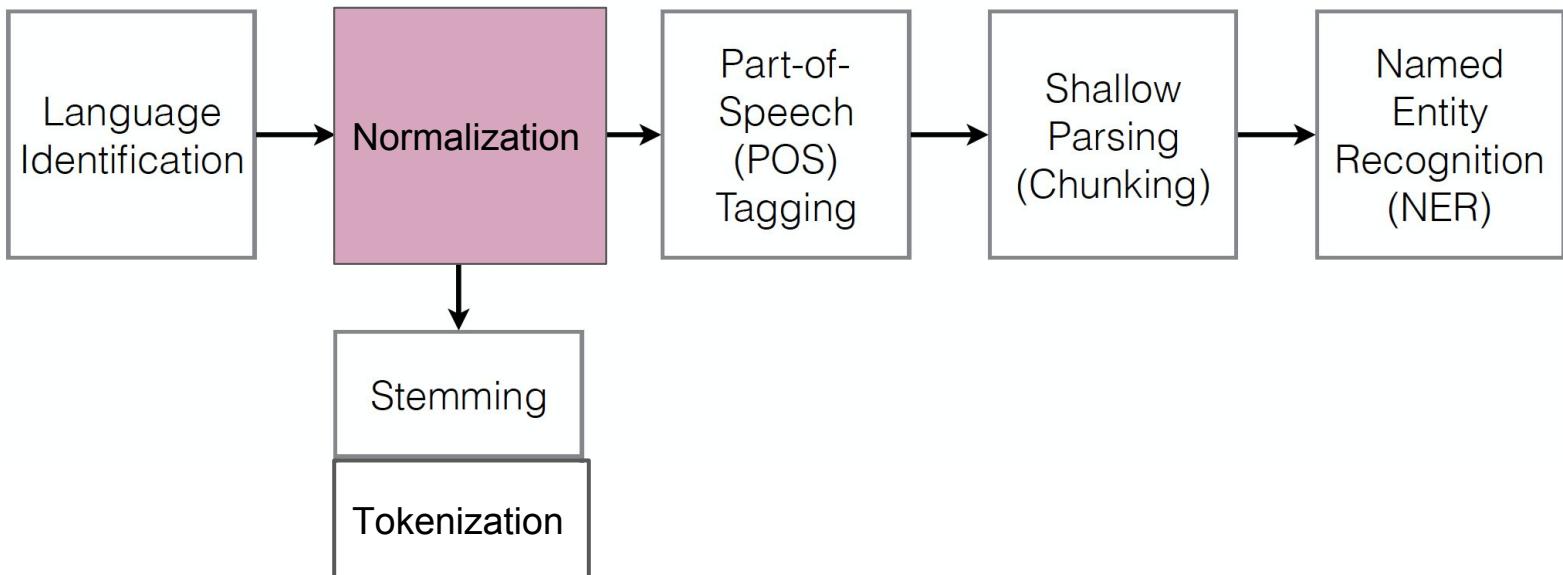
IM PRE-ORDERING THIS NOOOOOOW A.S.A.P!!!!!!

REPLY

1



# Typical NLP Pipeline



# 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

# Tokenization

Means taking an entire text stream and breaking into individual units (tokens) of analysis.

Why do we need to break the text into tokens?

Languages like English already include a word separator.

# Tokenization

White space is not sufficient.

We need to handle:

- Contractions: don't, couldn't → do not (or do n't), could not (could n't)
- Punctuation marks: Are you done? --? Are you done ?
  - But be careful with abbreviations: Mrs. Hebert

# How do we tokenize text?

- We use hand-crafted knowledge
  - regular expressions

# 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 |

# Regular Expressions: Negation in Disjunction

- Negations  $[^\wedge Ss]$ 
  - Carat means negation only when first in []

| Pattern              | Matches                  |                                      |
|----------------------|--------------------------|--------------------------------------|
| $[^\wedge A-Z]$      | Not an upper case letter | O <u>y</u> fn prিপেtchik             |
| $[^\wedge Ss]$       | Neither 'S' nor 's'      | I have no exquisi <b>t</b> e reason" |
| $[^\wedge e^\wedge]$ | Neither e nor ^          | Look h <u>ere</u>                    |
| $a^\wedge b$         | The pattern a carat b    | Look up <u>a^\wedge b</u> now        |

# Regular Expressions: More Disjunction

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

| Pattern                       | Matches       |
|-------------------------------|---------------|
| groundhog   woodchuck         |               |
| yours   mine                  | yours<br>mine |
| a   b   c                     | = [abc]       |
| [gG] roundhog   [Ww] oodchuck |               |



# Regular Expressions: ? \* + .

| Pattern | Matches                    |   |
|---------|----------------------------|---|
| colou?r | Optional previous char     | <u>color</u> <u>colour</u>                          |
| oo*h!   | 0 or more of previous char | <u>oh!</u> <u>ooh!</u> <u>oooh!</u> <u>ooooh!</u>   |
| o+h!    | 1 or more of previous char | <u>oh!</u> <u>ooh!</u> <u>oooh!</u> <u>ooooh!</u>   |
| baa+    |                            | <u>baa</u> <u>baaa</u> <u>aaaaa</u> <u>aaaaaa</u>   |
| beg.n   |                            | <u>begin</u> <u>bequn</u> <u>begun</u> <u>beq3n</u> |



Stephen C Kleene

Kleene \*, Kleene +

# Regular Expressions: Anchors ^ \$

| Pattern     | Matches                           |
|-------------|-----------------------------------|
| ^ [A-Z]     | Palo Alto                         |
| ^ [^A-Za-z] | 1 <u>"Hello"</u>                  |
| \. \$       | The end <u>.</u>                  |
| . \$        | The end <u>?</u> The end <u>!</u> |

# Example

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

# Errors

- The process we just went through was based on **fixing two kinds of errors**
  - Matching strings that we should not have matched (**cheat, features, meat**)
    - **False positives (Type I)**
  - Not matching things that we should have matched (**Eat**)
    - **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
  - REs are often the first model for any text processing task
- For many hard tasks, we use machine learning classifiers
  - But REs are used as features in the classifiers
  - Can be very useful in capturing generalizations

# **Basic Text Processing**

Word tokenization

# 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?

Night gathers, and now my watch begins. It shall not end until my death.

- **Type**: an element of the vocabulary.
- **Token**: an instance of that type in running text.
- How many?
  - 14? 17? tokens
  - 13? 15? types

# How many words?

$N$  = number of tokens

$V$  = vocabulary = set of types

$|V|$  is the size of the vocabulary

Herdan's law (1960):  $|V| > O(N^\beta)$   
 $\beta = 0.67 - 0.75$

|                                 | 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' < file.txt  
| sort  
| uniq -c
```

Change all non-alpha to  
newlines

Sort in alphabetical order

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' < text.txt | head
```

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

# The second step: sorting

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

A

A

A

A

A

A

A

A

A

...

# More counting

- Merging upper and lower case

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

- Sorting the counts

```
tr 'A-Z' 'a-z' < text.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., \$79.99, #newyearsresolutions → ??

# 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

- 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
  - 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

# **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 exempl compress and  
compress ar both accept  
as equival 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    | → Ø  | cats     | → cat    |

### Step 1b

|          |     |           |           |
|----------|-----|-----------|-----------|
| (*v*)ing | → Ø | walking   | → walk    |
|          |     | sing      | → sing    |
| (*v*)ed  | → Ø | plastered | → plaster |

### Step 2 (for long stems)

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

### Step 3 (for longer stems)

|      |     |            |          |
|------|-----|------------|----------|
| al   | → Ø | revival    | → reviv  |
| able | → Ø | adjustable | → adjust |
| ate  | → Ø | activate   | → activ  |
| ...  |     |            |          |

...

# Viewing morphology in a corpus

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

# Viewing morphology in a corpus

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

(\*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
      541 nothing
      388 king
      375 bring
      358 thing
      307 ring
      152 something
      145 coming
      130 morning
      122 having
      120 living
      117 loving
      116 Being
      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 require 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'

# Tokenization and Normalization

- tokenization: segmenting words in text
- normalization: mapping tokens to a standard format
- we use FSA to do most of the work
- each tokenizer will have its own rules and the end result will be different!
- preprocessing can make or break an experiment (Fokkens et al., 2013)

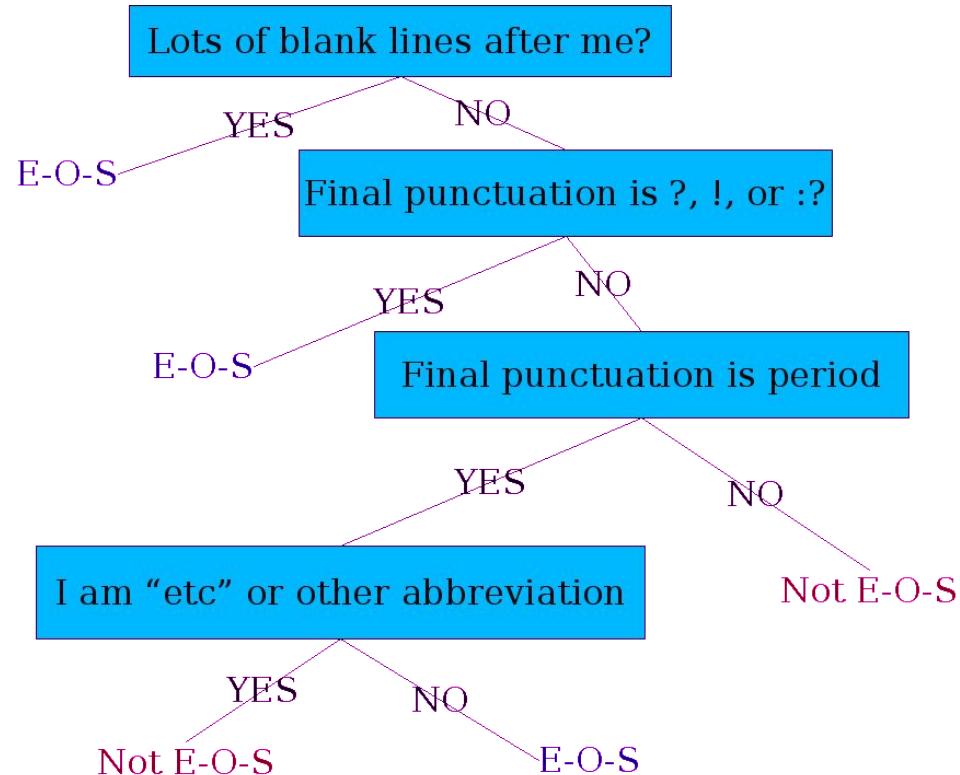
# **Basic Text Processing**

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.