Basic Text Processing

Regular Expressions

Word Tokenization

Word Normalization

Sentence Segmentation

Many slides adapted from slides by Dan Jurafsky

Basic Text Processing

Regular Expressions

Regular expressions

- A formal language for specifying text strings
- How can we search for any of these?
  - woodchuck
  - woodchuck**s**
  - **W**oodchuck
  - **W**oodchuck**s**

Regular Expressions: Disjunctions

Regular Expressions: Disjunctions

- Letters inside square brackets []

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

Regular Expressions: Disjunctions

- Letters inside square brackets []

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

Regular Expressions: Disjunctions

- Letters inside square brackets []

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

Regular Expressions: Disjunctions

- Letters inside square brackets []

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

Regular Expressions: Disjunctions

- Letters inside square brackets []

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

- Ranges `[A-Z]`

Pattern	Matches	the First Match in an example
`[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 `[^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
`groundhog|woodchuck`	
`yours|mine`	`yours` `mine`
`a|b|c|ab`	abc
`[gG]roundhog|[Ww]oodchuck`	

Regular Expressions: ? * + .

Pattern	Matches	
colou?r	0 or 1 of previous char	color colour
oo*h!	0 or more of previous char	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

Stephen C Kleene

Kleene *, Kleene +

Regular Expressions: Anchors ^ $

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

Example

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 task
- 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

Regular Expressions

Basic Text Processing

Word tokenization

Text Normalization

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

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

How many words?

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

- **Type**: an element of the vocabulary.

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 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?

How many words?

N = number of tokens

How many words?

N = number of tokens

V = vocabulary = set of types

$|V|$ is the size of the vocabulary

How many words?

N = number of tokens

V = vocabulary = set of types

$|V|$ is the size of the vocabulary

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

How many words?

N = number of tokens

V = vocabulary = set of types

$|V|$ is the size of the vocabulary

Church and Gale (1990): $|V| > O(N^{½})$

	Tokens = N	Types = |V|
Switchboard phone	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
```

tr: translate, -s: squeeze, -c: complement

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

Will likes to eat.

Will likes to laugh.

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

tr: translate, -s: squeeze, -c: complement

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

Will likes to eat.

Will likes to laugh.

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

tr: translate, -s: squeeze, -c: complement

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

Will likes to eat.
Will likes to laugh.

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 in alphabetical order

tr: translate, -s: squeeze, -c: complement

Will likes to eat.
Will likes to laugh.

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 in alphabetical order

tr: translate, -s: squeeze, -c: complement

Merge and count each type

Will likes to eat.
Will likes to laugh.

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
```

Change all non-alpha to newlines

Sort in alphabetical order

tr: translate, -s: squeeze, -c: complement

Merge and count each type

Will likes to eat.
Will likes to laugh.

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

tr: translate, -s: squeeze, -c: complement

Merge and count each type

Will likes to eat.
Will likes to laugh.

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

tr: translate, -s: squeeze, -c: complement

Merge and count each type

Will likes to eat.
Will likes to laugh.

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

tr: translate, -s: squeeze, -c: complement

Merge and count each type

```
1945 A
```

Will likes to eat.
Will likes to laugh.

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

tr: translate, -s: squeeze, -c: complement

Merge and count each type

```
1945 A
  72 AARON
```

Will likes to eat.
Will likes to laugh.

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

tr: translate, -s: squeeze, -c: complement

Merge and count each type

```
1945 A
  72 AARON
  19 ABBESS
```

Will likes to eat.

Will likes to laugh.

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

tr: translate, -s: squeeze, -c: complement

Merge and count each type

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

Will likes to eat.

Will likes to laugh.

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

tr: translate, -s: squeeze, -c: complement

Merge and count each type

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

Will likes to eat.

Will likes to laugh.

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

tr: translate, -s: squeeze, -c: complement

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
.... …
```

Will likes to eat.
Will likes to laugh.

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
```

tr: translate, -s: squeeze, -c: complement

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

Will likes to eat.
Will likes to babble.

1 babble
1 eat
2 likes
2 to
2 Will

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

tr: translate, -s: squeeze, -c: complement

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

Will likes to eat.
Will likes to babble.

1 babble
1 eat
2 likes
2 to
2 Will

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

tr: translate, -s: squeeze, -c: complement

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

Will likes to eat.
Will likes to babble.

1 babble
1 eat
2 likes
2 to
2 Will

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 in alphabetical order

tr: translate, -s: squeeze, -c: complement

Will likes to eat.
Will likes to babble.

1 babble
1 eat
2 likes
2 to
2 Will

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 in alphabetical order

tr: translate, -s: squeeze, -c: complement

Merge and count each type

Will likes to eat.
Will likes to babble.

1 babble
1 eat
2 likes
2 to
2 Will

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
```

Change all non-alpha to newlines

Sort in alphabetical order

tr: translate, -s: squeeze, -c: complement

Merge and count each type

Will likes to eat.
Will likes to babble.

1 babble
1 eat
2 likes
2 to
2 Will

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

tr: translate, -s: squeeze, -c: complement

Merge and count each type

Will likes to eat.
Will likes to babble.

1 babble
1 eat
2 likes
2 to
2 Will

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

tr: translate, -s: squeeze, -c: complement

Merge and count each type

Will likes to eat.
Will likes to babble.

1 babble
1 eat
2 likes
2 to
2 Will

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

tr: translate, -s: squeeze, -c: complement

Merge and count each type

```
1945 A
```

Will likes to eat.
Will likes to babble.

1 babble
1 eat
2 likes
2 to
2 Will

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

tr: translate, -s: squeeze, -c: complement

Merge and count each type

```
1945 A
  72 AARON
```

Will likes to eat.
Will likes to babble.

1 babble
1 eat
2 likes
2 to
2 Will

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

tr: translate, -s: squeeze, -c: complement

Merge and count each type

```
1945 A
  72 AARON
  19 ABBESS
```

Will likes to eat.
Will likes to babble.

1 babble
1 eat
2 likes
2 to
2 Will

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

tr: translate, -s: squeeze, -c: complement

Merge and count each type

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

Will likes to eat.
Will likes to babble.

1 babble
1 eat
2 likes
2 to
2 Will

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

tr: translate, -s: squeeze, -c: complement

Merge and count each type

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

Will likes to eat.
Will likes to babble.

1 babble
1 eat
2 likes
2 to
2 Will

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

tr: translate, -s: squeeze, -c: complement

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
.... …
```

Will likes to eat.
Will likes to babble.

1 babble
1 eat
2 likes
2 to
2 Will

The first step: tokenizing

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

(head: will print the first lines (10 by default) of its input. head -n NUM input)

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

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

More counting

- Merging upper and lower case

More counting

- Merging upper and lower case

```
tr ‘A-Z’ ‘a-z’ < shakes.txt | tr –sc ‘A-Za-z’ ‘\n’ | sort | uniq –c
```

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 (-n: numerical value, -k: column, -r: reverse)

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 (-n: numerical value, -k: column, -r: reverse)

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

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 (-n: numerical value, -k: column, -r: reverse)

```
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
```

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 (-n: numerical value, -k: column, -r: reverse)

```
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

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

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

Max-match segmentation illustration

- Thecatinthehat

Max-match segmentation illustration

- Thecatinthehat

the cat in the hat

Max-match segmentation illustration

- Thecatinthehat

the cat in the hat

- Thetabledownthere

Max-match segmentation illustration

- Thecatinthehat
- Thetabledownthere

the cat in the hat

the table down there

Max-match segmentation illustration

- Thecatinthehat
- Thetabledownthere

the cat in the hat

the table down there

theta bled own there

Max-match segmentation illustration

- Thecatinthehat
- Thetabledownthere

the cat in the hat

the table down there

theta bled own there

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!

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!

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

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 tokenization

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***

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

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***

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***

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* $\rightarrow$ *be*
  - *car, cars, car's, cars'* $\rightarrow$ *car*

Context dependent. for instance:
in our last meeting (noun, meeting).
We're meeting (verb, meet) tomorrow.

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*

Context dependent. for instance:
in our last meeting (noun, meeting).
We're meeting (verb, meet) tomorrow.

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

Context dependent. for instance:
in our last meeting (noun, meeting).
We're meeting (verb, meet) tomorrow.

Lemmatization

- Reduce inflections or variant forms to base form
  - *am, are, is* → *be*
  - *car, cars, car's, cars'* → *car*

Context dependent. for instance:
in our last meeting (noun, meeting).
We're meeting (verb, meet) tomorrow.

- *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

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

context independent

- Reduce terms to their stems in information retrieval

Stemming

context independent

- 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***.

Stemming

context independent

- 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.

Stemming

context independent

- 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

Porter's algorithm
The most common English stemmer

fixed rules put in groups, applied in order. https://tartarus.org/martin/PorterStemmer/

Text

Porter's algorithm
The most common English stemmer

fixed rules put in groups, applied in order. https://tartarus.org/martin/PorterStemmer/

Step 1a

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

Text

Porter's algorithm
The most common English stemmer

fixed rules put in groups, applied in order. https://tartarus.org/martin/PorterStemmer/

Step 1a

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

Text

Step 1b

```
(*v*)ing → ø  walking   → walk
              sing      → sing
(*v*)ed  → ø  plastered → plaster
…
```

Porter's algorithm
The most common English stemmer

fixed rules put in groups, applied in order. https://tartarus.org/martin/PorterStemmer/

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
…
```

Text

Porter's algorithm
The most common English stemmer

fixed rules put in groups, applied in order. https://tartarus.org/martin/PorterStemmer/

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
…
```

Text

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`

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
```

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
```

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
```

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
```

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

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
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

Word Normalization and Stemming

Basic Text Processing

Sentence Segmentation and Decision Trees

Sentence Segmentation

Sentence Segmentation

- !, ? are relatively unambiguous

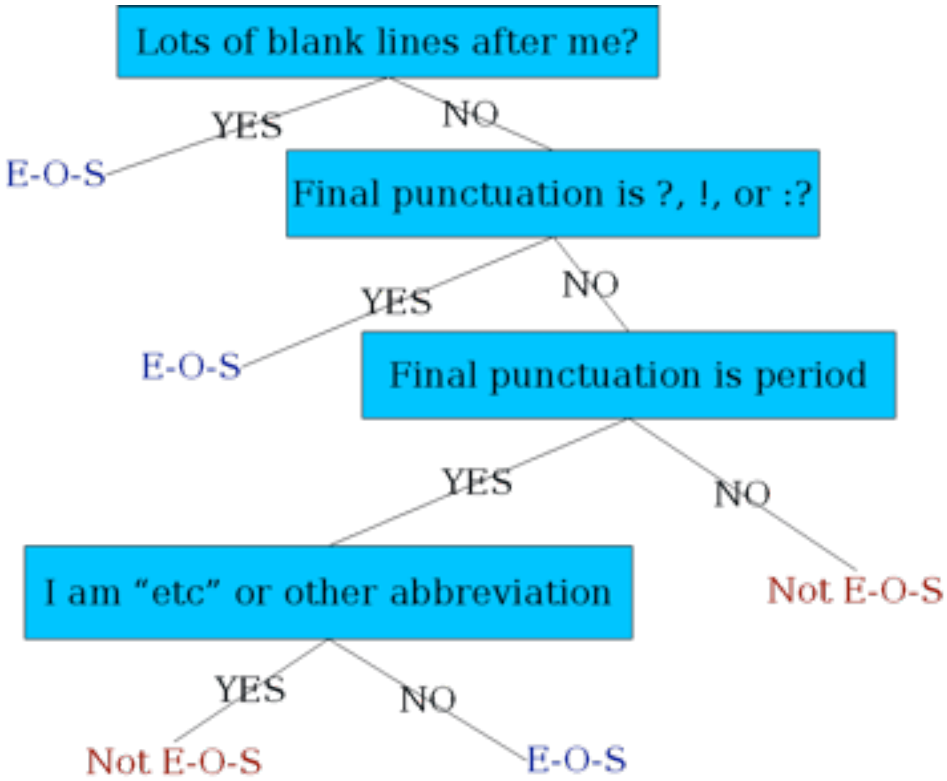
Sentence Segmentation

- !, ? are relatively unambiguous
- Period "." is quite ambiguous
  - Sentence boundary
  - Abbreviations like Inc. or Dr.
  - Numbers like .02% or 4.3

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

More sophisticated decision tree features

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

More sophisticated decision tree features

- Case of word with ".": Upper, Lower, Cap, Number
- Case of word after ".": Upper, Lower, Cap, Number

More sophisticated decision tree features

- Case of word with ".": Upper, Lower, Cap, Number
- Case of word after ".": Upper, Lower, Cap, Number

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

Implementing Decision Trees

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

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

Decision Trees and other classifiers

- We can think of the questions in a decision tree

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.

Sentence Splitters

- Stanford coreNLP: (deterministic)
- http://stanfordnlp.github.io/CoreNLP/

- UIUC sentence splitter: (deterministic)
- https://cogcomp.cs.illinois.edu/page/tools_view/2

Basic Text Processing

Sentence Segmentation and Decision Trees