Example “the”

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

/the/

Misses capitalized examples

/[tT]he/

won’t treat underscores and numbers as word (the_ or the25)

/\b[tT]he\b/

Incorrectly returns other or theology

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

It is not being mentioned that “the” word should be in beginning of a line.

Example “the”

There are also two other anchors:

- \b matches a word boundary
- \B matches a non-boundary

For example, /\b99\b/ will match the string 99 in

“There are 99 bottles of beer on the wall”

(because 99 follows a space)

But not 99 in

“There are 299 bottles of beer on the wall”

(since 99 follows a number).

But it will match 99 in $99

(since 99 follows a dollar sign ($), which is not a digit, underscore, or letter).

Example “the”

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

“the” word should be in beginning of a line or in end of a line

/(^|[^a-zA-Z])[tT]he([^a-zA-Z]|$)/

Example “the”

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, the)

False negatives (Type II)

Example “the”

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).

Complex Example

Let's try out a more significant example of the RE.

For example; build an application to help a user buy a computer on the Web.

- The user might want

"any machine with more than 6 GHz and 500 GB of disk space for less than $1000".

- To do this kind of retrieval, initially analyze expressions like 6 GHz or 500 GB or Mac or $999.99.

In the rest of the section some simple regular expressions will be analyzed for this task.

Complex Example

First, let's complete RE for prices.

RE for a dollar sign followed by a string of digits:
/$[0-9]+/

$ is for the end

Decimal point and two digits afterwards
/$[0-9]+\.[0-9][0-9]/

This pattern only allows $199.99 but not $199

Complex Example

Make the cents optional and to make sure a word boundary:

/\b$[0-9]+(\.[0-9][0-9])?\b/

How about specifications for processor speed?

Here's a pattern for that:

/\b[0-9]+␣*(GHz|[Gg]igahertz)\b/

Allowing optional fractions again *"5.5 GB"*;

/\b[0-9]+(\.[0-9]+)? *(GB|[Gg]igabytes)?\b/

␣ means White Spaces

Regular Expression

RE /{3}/ means
“exactly 3 occurrences of the previous character or expression”.

Therefore,
/a\.{24}z/
will match ***a*** followed by ***<u>24 dots</u>*** followed by ***<u>z</u>***

/a\.{24, 30}z/
will match ***a*** followed by ***<u>24 dots OR upto 30 dots</u>*** followed by ***<u>z</u>***

/a\.{24, }z/
will match ***a*** followed by **<u>at least *24 dots*</u>** followed by ***<u>z</u>***

Regular Expression

Substitutions

To do substitutions put parentheses (and) around the first pattern and use the number operator \1 in the second pattern to refer back.

Here's how it looks:

s/([0-9]+)/<\1>/

The \1 will be replaced by whatever string matches the first item in parentheses.

Therefore, this will match

The bigger they were, the bigger they will be

but not

The bigger they were, the faster they will be

Regular Expression

Capture group

Using parentheses to store a pattern in memory is called a capture group.

Every time a capture group is used (i.e., parentheses surround a pattern), the resulting match is stored in a numbered register.

For example; match two different sets of parentheses, \2 means whatever matched the second capture group.

Thus

/the (.*)er they (.*), the \1er we \2/

The faster they ran, the faster we ran

but not

The faster they ran, the faster we ate.

Similarly, the third capture group is stored in \3, the fourth is \4, and so on.

Regular Expression

Occasionally parentheses are used for grouping, but don't want to capture the resulting pattern in a register.

In that case, we use a **non-capturing group**, which is specified by putting the commands **?:** after the open parentheses, in the form (?: pattern).

For example;

/(?:some|a few) (people|cats) like some \1/

will match

some cats like some people

but not

some people like some a few

Hence the above example means that (?: some|a few) are not selected.

Exercise

The faster they ran, the faster we ran
The faster they walk, the faster we walk
The slower they ran, the slower we ran
The slower they walk, the slower we walk

How can you make a regular expression of above example?

s/The (.*) they (.*), the \1 we \2/

Exercise

The faster they ran, the slower we ran
The slower they ran, the faster we ran

How can you make a regular expression of above example?

Exercise

The faster they ran, the slower we ran
The slower they ran, the faster we ran

How can you make a regular expression of above example?

s/The (faster|slower) they (.*), the \1 we (.*)/

OR

s/The (faster|slower) they (.*), the \1 we \2/

OR

s/The (faster|slower) they ran, the \1 we ran/

Regular Expression (ELIZA)

Substitutions and capture groups are very useful in implementing simple chatbots like ELIZA

User 1:	Men are all alike.
ELIZA1:	**IN WHAT WAY**
User 2:	They're always bugging us about something or other.
ELIZA2:	**CAN YOU THINK OF A SPECIFIC EXAMPLE**
User 3:	Well, my friend made me come here.
ELIZA3:	**YOUR FRIEND MADE YOU COME HERE**
User 4:	He says I'm depressed much of the time.
ELIZA4:	**I AM SORRY TO HEAR YOU ARE DEPRESSED**

Regular Expression (ELIZA)

User 1:	Men are all alike.
ELIZA1:	**IN WHAT WAY**
User 2:	They're always bugging us about something or other.
ELIZA2:	**CAN YOU THINK OF A SPECIFIC EXAMPLE**
User 3:	Well, my friend made me come here.
ELIZA3:	**YOUR FRIEND MADE YOU COME HERE**
User 4:	He says I'm depressed much of the time.
ELIZA4:	**I AM SORRY TO HEAR YOU ARE DEPRESSED**

ELIZA works by having a series or cascade of regular expression substitutions each of which matches and changes some part of the input lines.

The first substitutions change all instances of ***my*** to ***YOUR***, and ***I'm*** to ***YOU ARE***, and so on.

Regular Expression (ELIZA)

User 1:	Men are all alike.
ELIZA1:	**IN WHAT WAY**
User 2:	They're always bugging us about something or other.
ELIZA2:	**CAN YOU THINK OF A SPECIFIC EXAMPLE**
User 3:	Well, my friend made me come here.
ELIZA3:	**YOUR FRIEND MADE YOU COME HERE**
User 4:	He says I'm depressed much of the time.
ELIZA4:	**I AM SORRY TO HEAR YOU ARE DEPRESSED**

```
s/.* all .*/IN WHAT WAY/
s/.* always .*/CAN YOU THINK OF A SPECIFIC EXAMPLE/
s/.* I'[mM] (depressed|sad) .*/I AM SORRY TO HEAR YOU ARE \1/
s/.* I AM (depressed|sad) .*/WHY DO YOU THINK YOU ARE \1/
```

Regular Expression (ELIZA)

User 1:	Men are all alike.
ELIZA1:	**IN WHAT WAY**
User 2:	They're always bugging us about something or other.
ELIZA2:	**CAN YOU THINK OF A SPECIFIC EXAMPLE**
User 3:	Well, my friend made me come here.
ELIZA3:	**YOUR FRIEND MADE YOU COME HERE**
User 4:	He says I'm depressed much of the time.
ELIZA4:	**I AM SORRY TO HEAR YOU ARE DEPRESSED**

What will be the RE of given sentence in selected dialog?

1) s/(my|your) friend made (me|you) come here/\1 friend made \2 come here/
2) s/(?: .*) (my|your) (.*) (me|you) (.*)/\1 \2 \3 \4/

Now more examples:

User 6: Ahhh, my wrist watch made me crazy.

User 7: Hmm, my car drives me cool.

Regular Expression

Lookahead

Finally, there will be times to predict the future: **lookahead** in the text to see

- if some pattern matches,
- but not advance the match cursor,
- Therefore, the pattern should occur

These **lookahead** assertions make use of the "?" syntax that check in the previous section for non-capture groups.

(?= pattern) is true if pattern occurs, but is **zero-width**, i.e. the match pointer doesn't advance the cursor.

(?! pattern) only returns true if a pattern does not match, but again is **zero-width** and doesn't advance the cursor.

Regular Expression

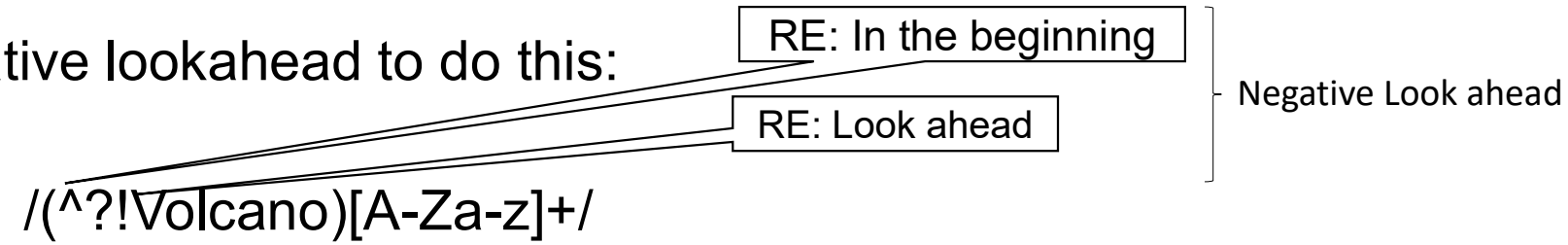
Negative Look ahead

Negative lookahead is commonly used when parsing some complex pattern but want to rule out a special case.

For example;

to match, at the beginning of a line, any single word that doesn't start with *"Volcano"*.

Negative lookahead to do this:



Exercise

Look ahead:

How to write a RE, if to match a word “But”, at the beginning of a line?

Solution: /(^?=But)[A-Za-z]+/

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, machine learning classifiers are used

- But regular expressions are used as features in the classifiers
- Can be very useful in capturing generalizations

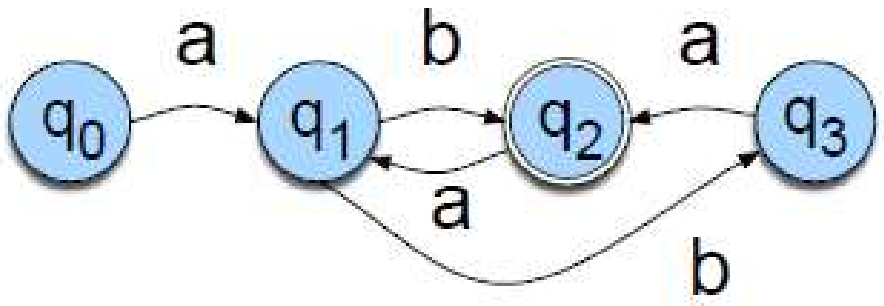
Exercise

Write regular expressions for the following languages.

1. the set of all alphabetic strings;
   /[A-Za-z]+/
2. the set of all lower case alphabetic strings ending in a "***b***";
   /[a-z]+b/
3. the set of all strings from the alphabet "***a***, ***b***" such that each ***a*** is immediately preceded by ***b***s and immediately followed by a ***b***;
   /b+ab/
   OR
   /bb*ab/

Exercise

Write the regular expression for the following FSA

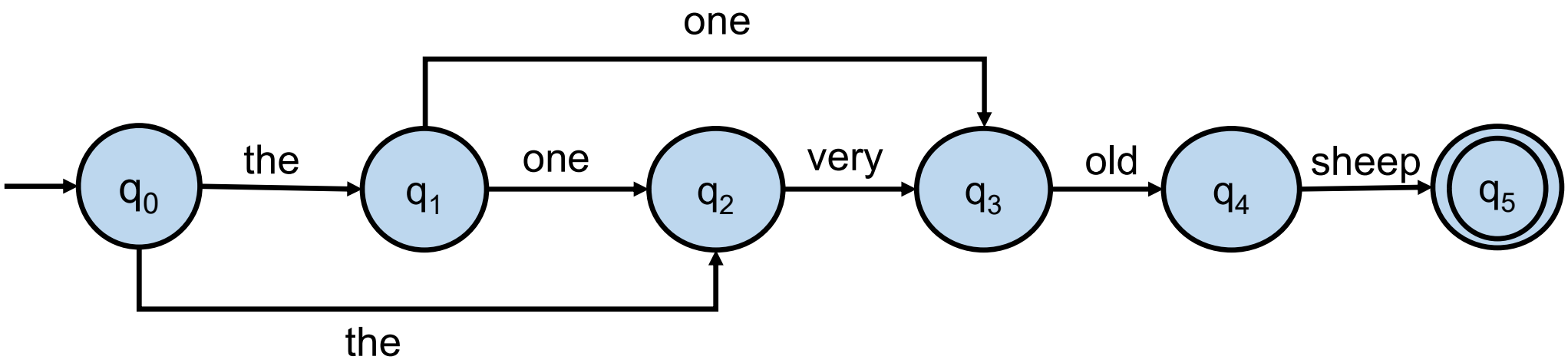


Solution:

/aba(ab)* | ab(ab)*/

Exercise

Write the set of Strings for the following FSA



Solution:

the one very old sheep
the very old sheep
the one old sheep

Basic Text Processing

Word tokenization

Words and Corpora

Corpus (plural = Corpora)

A collection of written or spoken material stored on a computer and used to find out how language is used.

Example:

He stepped out into the hall, was delighted to encounter a water brother.

This sentence has

- 13 words if we don't count punctuation marks as words,
- 15 if we count punctuation.

  Whether we treat period (“.”), comma (“,”), and so on as words depends on the task.

Punctuation is critical for finding boundaries of things (commas, periods, colons) and for identifying some aspects of meaning (question marks, exclamation marks, quotation marks).

Words and Corpora

Spoken Sentences

For example:

An utterance is the spoken correlate of a sentence:

I do uh main- mainly business data processing

The given is the spoken sentence (i.e. utterance) and has two kinds of disfluencies.

1. The broken-off word *main-* is called a *fragment*.
2. Words like *uh* and *um* are called *fillers* or *filled pauses*.

Should we consider these to be words?

Again, it depends on the application.

If we are building a speech transcription system, we might want to eventually strip out the disfluencies.

Words and Corpora

What is the reason to use fillers and fragments in spoken speech?

Disfluencies like *uh* or *um* are actually helpful in speech recognition in predicting the upcoming word,

because they may signal that the speaker is restarting the clause or idea,

Therefore for speech recognition they are treated as regular words.

Words and Corpora

How about inflected forms like *cats* versus *cat*?

These two words have the same lemma cat but are different wordforms.

Lemma

A lemma is a set of lexical forms having the same stem, the same major part-of-speech, and the same word sense.

Wordform

The wordform is the full inflected or derived form of the word.

For morphologically complex languages like Arabic, we often need to deal with lemmatization.

Words and Corpora

گردان فعل ماضی معروف

حال	جنس	واحد	تثنیہ	جمع
غائب	مذکر	فَعَلَ	فَعَلَا	فَعَلُوا
	مونث	فَعَلَتْ	فَعَلَتَا	فَعَلْنَ
حاضر	مذکر	فَعَلْتَ	فَعَلْتُمَا	فَعَلْتُمْ
	مونث	فَعَلْتِ	فَعَلْتُمَا	فَعَلْتُنَّ
متکلم	مذکر و مونث	فَعَلْتُ	فَعَلْنَا	

Words and Corpora

For many tasks in English, however, **wordforms** are sufficient

How many wordforms are there in English?

To answer this question we need to word type distinguish two ways of talking about words.

Types are the number of distinct words in a corpus; if the set of words in the vocabulary is V, the number of types is the word token vocabulary size |V|.

Tokens are the total number N of running words.

How many words?

I do uh main- mainly business data processing

Fragments (i.e. main-), **filled pauses** (i.e. uh)

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 Tokens or types in given example?

- 15 tokens (or 14)

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

- 13 types (or 12) (or 11?)

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

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

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

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?

N = number of tokens

V = vocabulary = set of types

|V| is the size of the vocabulary

Heap's Law: $V_R(n) = kn^{\beta}$

Heaps' law (also called **Herdan's law**) describes the number of distinct words in a document (or set of documents) as a function of the document length (so called type-token relation).

The formula is given:

1) V_R is the number of distinct words in an instance text of size n.
2) k and β are free parameters determined various experiments.

With English text corpora, typically k is between 10 and 100, and β is between 0.4 and 0.6.

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

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.	→	??
555,550.50	→	??
rock 'n' roll	→	rock and roll (separate tokens), one token?

Issues in Tokenization

A tokenizer can also be used to expand **clitic** contractions that are marked by apostrophes, for example,

- what're to the two tokens what are,
- we're to the two tokens we are.

A **clitic** is a part of a word that can't stand on its own and can only occur when it is attached to another word.

Some such contractions occur in other alphabetic languages, including articles and pronouns in French (j'ai, l'homme).

Depending on the application, tokenization algorithms may also tokenize multiword expressions for example:

New York or rock 'n' roll as a single token, which requires a multiword expression dictionary of some sort.

Tokenization is thus intimately tied up with named entity detection, the task of detecting names, dates, and organizations

Issues in Tokenization

Penn Treebank tokenization

Penn Treebank tokenization standard, used for the parsed corpora (treebanks) released by the Linguistic Data Consortium (LDC), the source of many useful datasets.

This standard separates out

- **clitics** (i.e. *doesn't* becomes *does* plus *n't*)
- keeps hyphenated words together
- and separates out all punctuation

Input	"	The	San	Francisco-based	restaurant	,	"	they	said	,	"	does	n't	charge	$	10	"	.
Output	"	The	San	Francisco-based	restaurant	,	"	they	said	,	"	does	n't	charge	$	10	"	.

Tokenization: Language Issues

French

L'ensemble → one token or two?

L'ensemble => means “the entire”

- ***L*** ? ***L'*** ? ***Le*** ?
- Want ***l'ensemble*** to match with ***un ensemble***

German noun compounds are not segmented

Lebensversicherungsgesellschaftsangestellter

Lebens versicherungs gesellschafts angestellter

means ‘life insurance company employee’

German information retrieval needs **compound splitter**

Tokenization: Language Issues

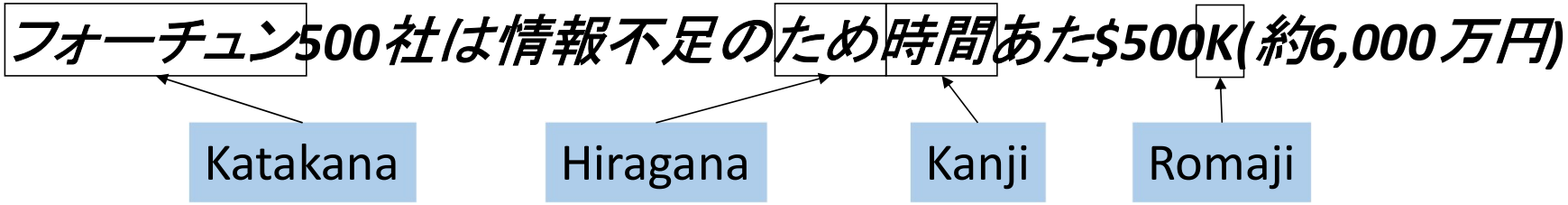
Chinese and Japanese no spaces between words:

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



End-user can express query entirely in hiragana!

Word Tokenization in Chinese

The example provided in previous slides are 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 or Max-Match)

Maximum Matching Word Segmentation Algorithm

Given a wordlist of any string as dictionary, 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
    1) Move the pointer over the word in string
    2) Go to 2

Maximum Matching Word Segmentation Algorithm

```
function MAXMATCH(sentence, dictionary D) returns word sequence W
        if sentence is empty
                return empty list
        for i length(sentence) downto 1
                firstword = first i chars of sentence
                remainder = rest of sentence
                if InDictionary(firstword, D)
                        return list(firstword, MaxMatch(remainder,dictionary) )
        # no word was found, so make a one-character word
        firstword = first char of sentence
        remainder = rest of sentence
return list (firstword, MaxMatch(remainder,dictionary D) )
```

Max-Match Segmentation

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 Language

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

Modern probabilistic segmentation algorithms even better

Max-Match Segmentation

Also works well in German Language

Lebensversicherungsgesellschaftsangestellter

Lebens	*versicherungs*	*gesellschafts*	*angestellter*
life	insurance	company	employee

Basic Text Processing

Word Normalization and Stemming

Normalization

Tokens can also be Normalized, and choose the Normalized form to match multiple forms

Information Retrieval: indexed text & query terms must have same form.

For example to match **U.S.A.** to **USA** and **US**

or to match **uh-huh** to **uhhuh**

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

Case folding is another form of Normalization.

For example: everything is matched to lower case.

Applications like Information Retrieval: reduce all letters to lower case

- Normally 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 is used for

- Sentiment Analysis,
- Machine Translation,
- Information extraction

Case folding is helpful (US versus us is important)

Lemmatization

Lemmatization: determines that two words have the same root, despite their surface differences

Lemmatization is used to reduce 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

How Lemmatization is performed?

Through **Morphological Parsing**

Morphology is the way to build Morphemes:

Morphemes means “The small meaningful units that make up words”

Two types of Morphemes are:

- Stems: The core meaning-bearing units
- Affixes: Adding additional meaning of various kinds OR 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 equival to compress

For stemming widely used algorithm is Porter's Algorithm

Porter's Algorithm

The most common English stemmer.

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.

The Porter stemmer algorithm produces the following stemmer output.

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'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
  - singing → 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
- …

Morphology in a Corpus

Why only strip –ing if there is a vowel?

(*v*)ing → ø

walking → walk

singing → sing

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

```
1312 King
 548 being
 541 nothing
 388 king
 375 bring
 358 thing
 307 ring
 152 something
 145 coming
 130 morning
```

```
 548 being
 541 nothing
 152 something
 145 coming
 130 morning
 122 having
 120 living
 117 loving
 116 Being
 102 going
```

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

Sentence segmentation is another important step in text processing.

The most useful cues for segmenting a text into sentences are punctuation, like periods, question marks, exclamation points.

Question marks, exclamation points and Periods are relatively unambiguous markers of sentence boundaries.

!, ? are relatively unambiguous

Sentence Segmentation

!, ? are relatively unambiguous

Period "." is quite ambiguous

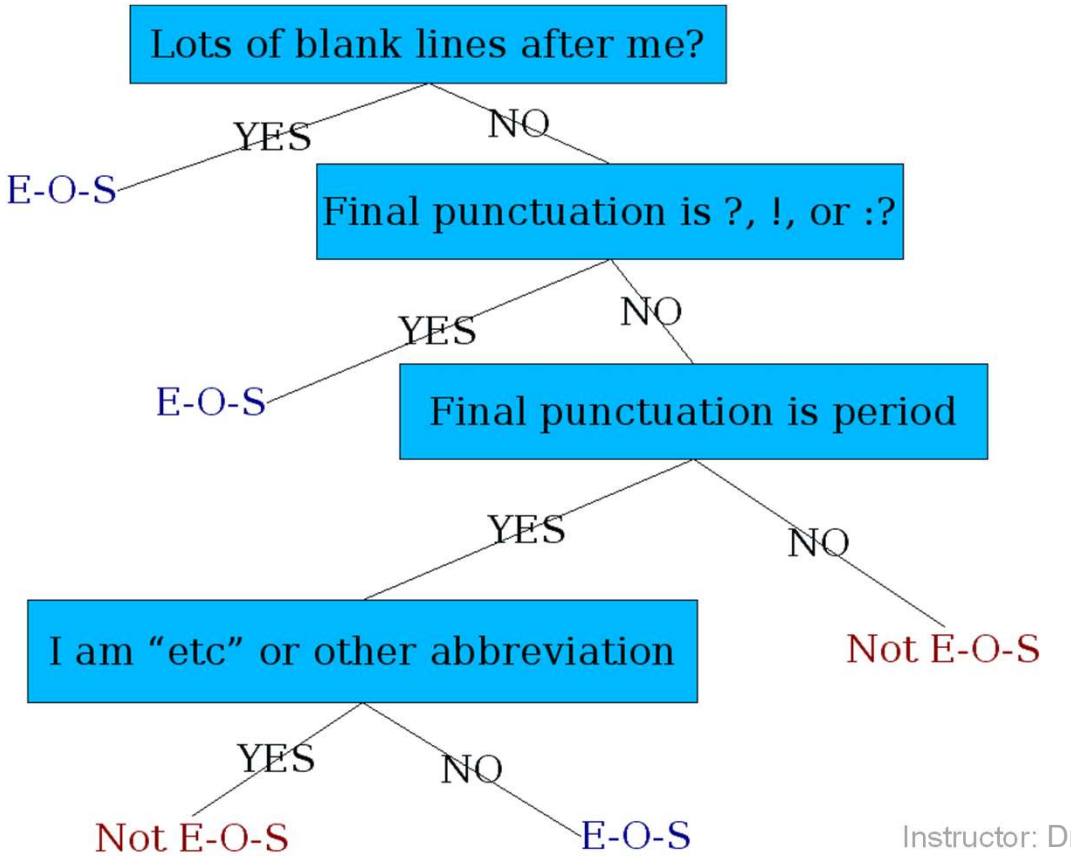
- Sentence boundary
- Abbreviations like Inc. or Dr.
- Numbers like .02% or 4.3

Sentence Segmentation/Tokenization works by:

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

For example: Dr. , Ph.D. , 50.5 , …

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

Q&A

That’s all for today’s Lecture