Regular Expressions: Overview

School of Information Studies
Syracuse University

Overview

Regular expressions (a.k.a. regex, regexp, or RE) are essentially a tiny, highly specialized programming language.

- Embedded inside Python, Perl, Java, php, and other languages

You can use this little language to specify the rules for a pattern to match any set of possible strings.

- Sentences, e-mail addresses, ads, dialogs, etc.

“Does this string match the pattern?” or “Is there a match for the pattern anywhere in this string?”

Regular expressions can also be used as a language generator; regular expression languages are the first in the Chomsky hierarchy.

Useful for Matching Text

A language for specifying patterns in text

Examples:

- Matching names like "Jane Q. Public"):
  /\b[A-Z][a-z]+ +[A-Z]\. +[A-Z][a-z]+\b/
- Matching all email addresses, with patterns like:
  …....@.....edu
  …....@.....gov
  ..….@.....com
- Matching all URLs
  - A fairly predictable set of characters and symbols selected from a finite set, (e.g., a-z, www, http, ~, /)
- And many others!

In these slides, we use the (Perl) convention that regular expressions are surrounded by / —Python uses " (quote mark)

Regular Expressions as a Formal Language

In language theory, regular expressions specify a language that can be recognized by finite-state automata (a.k.a. finite automaton, finite-state machine, FSA, or FSM).

- An abstract machine that can be used to implement regular expressions
  - Has a finite number of states, and a finite amount of memory (i.e., the current state)
- Can be represented by directed graphs or transition tables

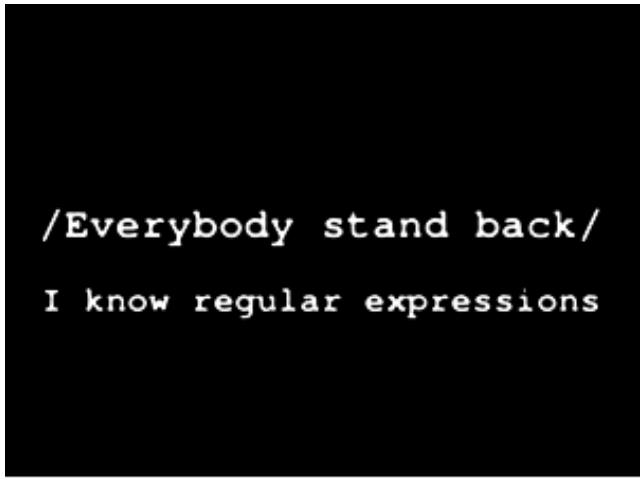
The regular languages are the first in the Chomsky hierarchy (context-free languages and context-sensitive languages are the next).

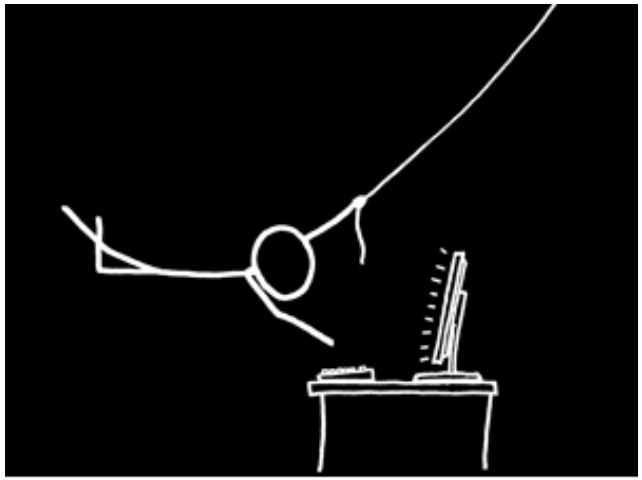
Regular languages are exactly the set of languages recognized by finite automata.

Regular Expressions Save the Day!

Regular expressions are a widely known and useful tool.

http://xkcd.com/208/





Regular Expressions: Basics

School of Information Studies
Syracuse University

Matching Text

Suppose that we have some text and we want to match any form of the word *woodchuck*.

- woodchuck
- Woodchuck
- woodchucks
- Woodchucks

Example word and picture from Dan Jurafsky

Character Classes

To exactly match a piece of text, the pattern is the text itself.

Pattern	Matches	Example
woodchuck	woodchuck	The **woodchuck** eats lettuce

If we want to match alternative characters, we put those characters inside square brackets to show a character class.

Pattern	Matches	Example
[wW]oodchuck	Woodchuck, woodchuck	**Woodchuck**
[abc]	Any a, b, or c	the woodsm**a**n
[1234567890]	Any digit	Any **3** of them, galaxy**2**

Character Class Ranges

Some larger character classes can be expressed as ranges. For example, read the first pattern below as "from a to z."

Pattern	Matches	Example
[a-z]	Any lower case letter	k
[A-Z]	Any upper case letter	**F**ourth of **J**uly
[0-9]	Any digit	Any **3** of them, galaxy**2**

Negation in Character Classes

To specify matching any character not listed in the class, put the ^ sign (read “carat” or “hat”) at the beginning of the class.

Pattern	Matches	Example
[^a-z]	Not a lowercase letter	**C**hapter **3** :
[^Zz]	Not Z or z	Zzzzzzz**o**zzz
[^s^]	Not s or ^	**Base ^ exp**
a^b	The pattern a^b	Look up **a^b** now

If a ^ character does not occur in the first position, then it just means the character itself.

Disjunction

Use the pipe character "|" to make a pattern that matches either the left or right of the |.

Pattern	Matches	Example
woodchuck|groundhog	Either word	A **woodchuck** is the same as a **groundhog**!
high|low	Either word	**high** tide, **low** tide
[gG]roundhog|[Ww]oodchuck		

The pipe operator can be combined with character classes and other operators.

Repetition Operators and “.”

Pattern	Matches	Example
colou?r	Optional previous character	**color colour**
o*h!	0 or more of previous character	**h! oh! ooh! ooooh!**
o+h!	1 or more of previous character	**oh! ooh! oooh! ooooh!**
beg.n	“.” matches any character	**begin begun began beg3n**

The “.” (read dot or period) is a special operator that allows you to match any character, sometimes referred to as a “wildcard.”

Anchor Tags: ^ and $

If the ^ appears at the beginning of the regular expression itself (not in a character class), then it means to match whatever follows only if it is at the beginning of the text string.

And $ means only match at the end of the text.

Pattern	Matches	Example
^[A-Z]	Any capital letter at the beginning	**P**alo Alto
^[^A-Za-z]	Any character not a letter at the beginning	**1** "Hello"
\.$	A dot at the end	The end**.**
.$	Any character at the end	The end**?** The en**d**

Regular Expressions: Notation Details

School of Information Studies
Syracuse University

Basic Notation Summary

Adding to our previous notations, we note that repetitions can be controlled by counters to give the exact number of repeats.

1. /**[abc]**/ = /a|b|c/ Character class; disjunction matches one of a, b, or c
2. /**[b-e]**/ = /b|c|d|e/ Range in a character class
3. /**[^b-e]**/ Complement of character class
4. /**.**/ Wildcard matches any character
5. /**a***/ /[af]*/ /(abc)*/ Kleene star: zero or more
6. /**a?**/ /(ab|ca)?/ Optional: zero or one;
7. /**a+**/ /([a-zA-Z]1|ca)+/ Kleene plus: one or more
8. /a{8}/ /b{1,2}/ /c{3,}/ Counters: exact number of repeats

Anchors, Grouping

Anchors

- Constrain the position(s) at which a pattern may match
- /^a/ Pattern must match at beginning of string
- /a$/ Pattern must match at end of string
- /\bword23\b/ "Word" boundary: /[a-zA-Z0-9_][^a-zA-Z0-9_]/ following /[^a-zA-Z0-9_][a-zA-Z0-9_]/
- /\B23\B/ "Word" **non**-boundary

Parentheses

- Can be used to *group* together parts of the regular expression, sometimes also called a *sub-match*

Regular Expressions

Escapes

- A backslash "\" placed before a character is said to "escape" (or "quote") the character. There are six classes of escapes:
  1. Numeric character representation: the octal or hexadecimal position in a character set: "\012" = "\xA"
  2. **Meta-characters: the characters that are syntactically meaningful to regular expressions and therefore must be escaped in order to represent themselves in the alphabet of the regular expression: "[] () {} | ^ $. ? + * \" (note the inclusion of the backslash)**
  3. "Special" escapes (from the "C" language):
     newline: "\n" = "\xA" carriage return: "\r" = "\xD"
     tab: "\t" = "\x9" formfeed: "\f" = "\xC"

Regular Expressions

Escapes (cont.)

4. **Aliases:** shortcuts for commonly used character classes (note that the capitalized version of these aliases refer to the complement of the alias's character class):
   - Whitespace: "**\s**" = "[\t\r\n\f\v]"
   - Digit: "**\d**" = "[0-9]"
   - Word: "**\w**" = "[a-zA-Z0-9_]"
   - Non-whitespace: "**\S**" = "[^ \t\r\n\f]"
   - Non-digit: "**\D**" = "[^0-9]"
   - Non-word: "**\W**" = "[^a-zA-Z0-9_]"
5. **Memory/registers/back-references:** "\1", "\2", etc.
6. **Self-escapes:** any characters other than those that have special meaning can be escaped, but the escaping has no effect; the character still represents the regular language of the character itself

Greediness

Regular expression counters/quantifiers that allow for a regular language to match a variable number of times (i.e., the star, the plus, "?", "{*min*,*max*}", and "{*min*,}") are inherently *greedy*.

- That is, when they are applied, they will match as many times as possible, up to *max* times in the case of "{*min*,*max*}", at most once in the "?" case, and infinitely many times in the other cases.
- Each of these quantifiers may be applied non-greedily by placing a question mark after it. Non-greedy quantifiers will at first match the **minimum** number of times.
- For example, against the string "From each according to his abilities":
  - /\b\w+.*\b\w+/ matches the entire string
  - /\b\w+.*?\b\w+/ matches just "From each"

Regular Expression Functions

School of Information Studies
Syracuse University

How to Use Regular Expressions

To make regular expressions useful, we want to ask:

- If a match occurs
- Which text was matched

In most languages with regular expressions, including Python (and Java), the regular expression is first defined with the compile function

pattern = re.compile("<regular expr>")

Then the pattern can be used to match strings

m = pattern.search(string)

where m will be true if the pattern matches anywhere in the string.

Another option is to use the function

re.match("<regular expr>", string)

which combines compile with the match function (next page).

Regular Expression Functions

Python includes other useful functions

- pattern.match—true if matches the beginning of the string
- pattern.search—scans through the string and is true if the match occurs in any position
  - These functions return a "MatchObject" or None if no match found
- pattern.findall—finds all occurrences of text that match and returns them in a list

RE Pattern MatchObjects

MatchObjects also have functions to find the matched text

- match.group ( )—returns the string(s) matched by the RE
  - Includes all the subgroups indicated by internal parentheses
- match.start ( )—returns the starting position of the match
- match.end ( )—returns the ending position of the match
- match.span ( )—returns a tuple containing the start, end
- And note that using the MatchObject as a condition in, for example, an If statement, will be true, while if the match failed, None will be false

Regular Expression Substitution

Once a regular expression has matched in a string, the matching sequence may be replaced with another sequence of zero or more characters.

- Convert "red" to "blue"
  - `p = re.compile("red") string = p.sub("blue", string)`
- Convert leading and/or trailing whitespace to an '=' sign:
  - ```
    p = re.compile("^\s+|\s+$")
    string = p.sub("=",string)
    ```
- Remove all numbers from the string: "These 16 cows produced 1,156 gallons of milk in the last 14 days."
  - ```
    p = re.compile(" \d{1,3}(,\d{3})*")
      string = p.sub("",string)
    ```
  - The result: "These cows produced gallons of milk in the last days."

Extensions to Regular Expressions

Memory/registers/back-references

- Many regular expression languages include a memory/register/back-reference feature, in which sub-matches may be referred to later in the regular expression, and/or when performing replacement, in the replacement string:
  A sub-match, or a match group, is defined as matching something in parentheses (as in the /(\w+)/), and the back-reference /\1/ says to match the same string that matched the sub-match.

```
p = re.compile("(\w+)\s+\1\b)")
p.search("Paris in the the spring").group()
returns 'the the'
```

If you want to use internal parentheses without triggering this feature, follow the left parenthesis with ?: to make a non-capturing subgroup:

```
p = recompile("\w+\s+(?:\w+\.)+")
```

Regular Expression Examples

Character classes and Kleene symbols

[A-Z]+ = one or more consecutive capital letters
matches GW or FA or CRASH

[A-Z]? = zero or one capital letter

So, [A-Z]ate

matches: Gate, Late, Pate, Fate; but not GATE or gate

and [A-Z]+ate

matches: Gate, GRate, Heate; but not Grate, or grate, or STATE

and [A-Z]*ate

matches: Gate, GRate, and ate; but not STATE, grate, or Plate

Regular Expression Examples

Some longer examples:

([A-Z][a-z]+)\s([a-z0-9]+)

matches: Intel c09yt745; but not IBM series5000

[A-Z]\w+\s\w+\s\w+[!]$

matches: The dog died!

But does not match “he said, “The dog died!” because the $ indicates end of line, and there is a quotation mark before the end of the line

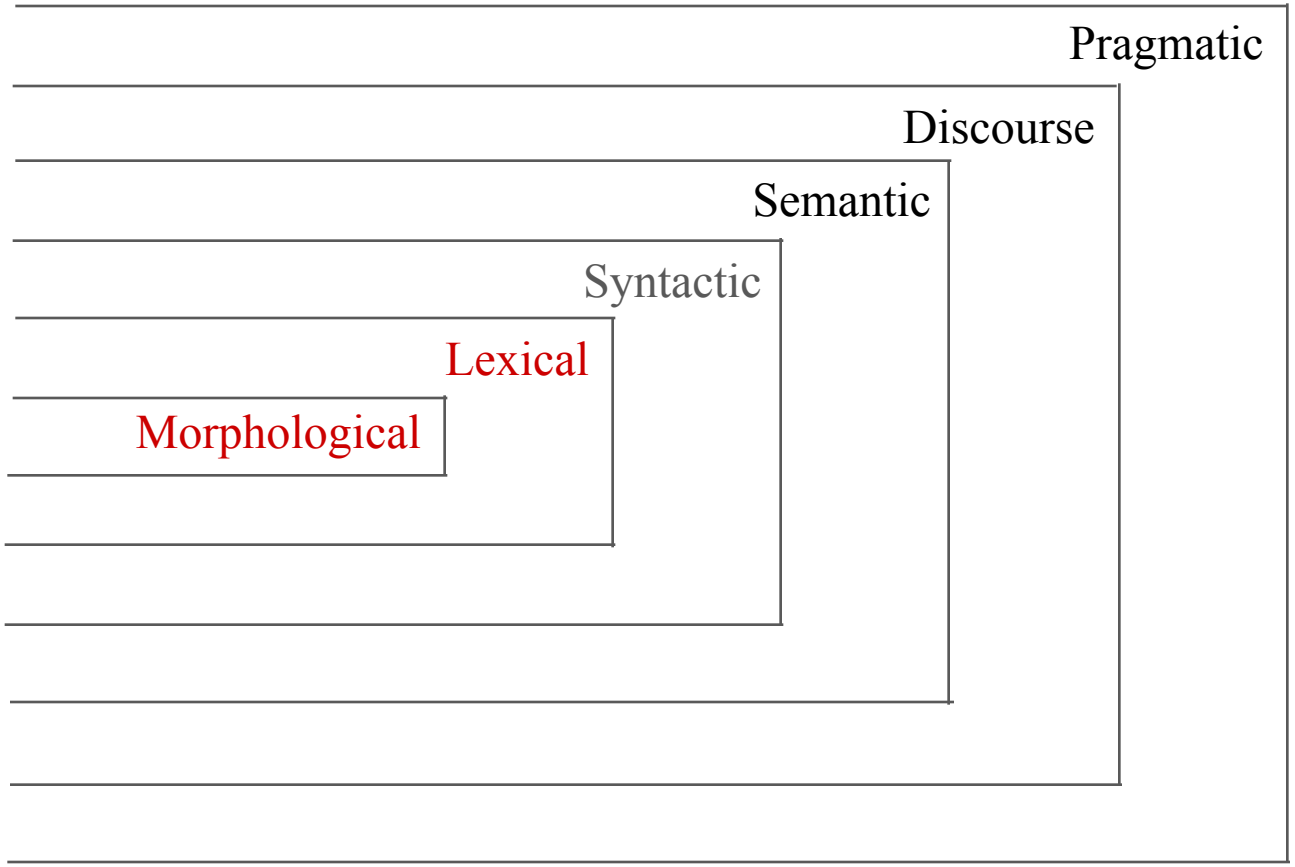
(\w+ats?\s)+

Parentheses define a pattern as a unit, so the above expression will match all the words in this string: Fat cats eat Bats that Splat

Morphology

School of Information Studies
Syracuse University

Levels of Language



Morphology

Morphology is the level of language that deals with the internal structure of words

General morphological theory applies to all languages as all natural human languages have systematic ways of structuring words (even sign language)

Must be distinguished from morphology of a specific language

- English words are structured differently from German words, although both languages are historically related
- Both are vastly different from Arabic

Minimal Units of Meaning

Morpheme = the minimal unit of meaning in a word

- walk
- -ed

Simple words cannot be broken down into smaller units of meaning

- Monomorphemes
- Called base words, roots, or stems

Affixes are attached to base words

- prefixes, infixes, suffixes, circumfixes

Affixes

Prefixes appear in front of the stem to which they attach

- un- + happy = unhappy

Infixes appear inside the stem to which they attach

- -blooming- + absolutely = absobloominglutely

Suffixes appear at the end of the stem to which they attach

- emotion = emote + -ion
- English may stack up to 4 or 5 suffixes to a word
- Agglutinative languages like Turkish may have up to 10

Circumfixes appear at both the beginning and end of stem

- German past participle of sagen is gesagt: ge- + sag + -t

Spelling and sound changes often occur at these boundaries in fusional languages, like English

- Very important for NLP

Inflection

Inflection modifies a word's form in order to mark the grammatical subclass to which it belongs

- apple (singular) > apples (plural)

Inflection does not change the grammatical category (part of speech)

- apple—noun; apples—still a noun

Inflection does not change the overall meaning

- Both *apple* and *apples* refer to the fruit

Derivation

Derivation creates a new word by changing the category and/or meaning of the base to which it applies

Derivation can change the grammatical category (part of speech)

- sing (verb) > singer (noun)

Derivation can change the meaning

- act of singing > one who sings

English has many derivational affixes, and they are regularly used to form new words

- Part of this is cultural
- English speakers readily accept newly introduced terms

-ation	computerize	computerization
-ee	appoint	appointee
-er	kill	killer
-ness	fuzzy	fuzziness

Inflection and Derivation: Order

Order is important when it comes to inflections and derivations

- Derivational suffixes must precede inflectional suffixes
  - sing + -er + -s is ok
  - sing + -s + -er is not
- This order may be used as a clue when working with natural language text

English has few inflections

- Many other languages use inflections to indicate the role of a word in the sentence
  - Use of case endings allows fairly free word order
- English instead has a fixed word order
  - Position in the sentence indicates the role of a word, so case endings are not necessary

Classes of Words

Closed classes are fixed—new words cannot be added

- Pronouns, prepositions, comparatives, conjunctions, determiners (articles and demonstratives)
- Function words

Open classes are not fixed—new words can be added

- Nouns, verbs, adjectives, adverbs
- Content words
- New content words are a constant issue for NLP

Word Formation Rules: Agreement

Plurals

- In English, the morpheme ‘s’ is often used to indicate plurals in nouns
- Nouns and verbs must agree in plurality

Gender: nouns, adjectives and sometimes verbs in many languages are marked for gender

- Two genders (masculine and feminine) in Romance languages like French, Spanish, Italian
- Three genders (masculine, feminine, and neutral) in Germanic and Slavic languages
- More are called noun classes—Bantu has up to 20
- Gender is sometimes explicitly marked on the word as a morpheme; but sometimes it is just a property of the word

Ambiguous Affixes

Some affixes are ambiguous:

- -er
  - Derivational: Agentive –er Verb + -er > Noun
  - Inflectional: Comparative –er Adjective + -er > Adjective
- -s or –es
  - Inflectional: Plural Noun + -(e)s > Noun
  - Inflectional: Third person singular Verb + -(e)s > Verb
- -ing
  - Inflectional Progressive Verb + -ing > Verb
  - Derivational "act of" Verb + -ing > Noun
  - Derivational "in process of" Verb + -ing > Adjective

As with all other ambiguity in language, this morphological ambiguity creates a problem for NLP

Complex Morphology

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’

Stemming

School of Information Studies
Syracuse University

Stemming

Removal of affixes (usually suffixes) to arrive at a base form that may or may not necessarily constitute an actual word

Continuum from very conservative to very liberal modes of stemming

- Very conservative
  - Remove only plural *-s*
- Very liberal
  - Remove all recognized prefixes and suffixes

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 Stemmer

Popular stemmer based on work done by Martin Porter

- Porter, M. F. (1980). An algorithm for suffix stripping. *Program 14*(3), pp. 130–137.

Very liberal step stemmer with five steps applied in sequence

- See example rules on next slide.

Probably the most widely used stemmer

Does not require a lexicon

Open source software available for almost all programming languages

Examples of Porter Stemmer Rules

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

…

Where *v* is the occurrence of any verb

From Dan Jurafsky

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

…

Lemmatization

Removal of affixes (typically suffixes)

But the goal is to find a base form that does constitute an actual word

Example:

- *parties* → remove *-es*, correct spelling of remaining form *parti* → *party*

Spelling corrections are often rule-based

May use a lexicon to find actual words

Text Processing

School of Information Studies
Syracuse University

Basic Text Processing

Every NLP task needs to do text normalization to determine what are the words of the document.

- Segmenting/tokenizing words in running text
  - Special characters like hyphen (-) and apostrophe (’)
- Normalizing word formats
  - (Non-)capitalization of words
  - Reducing words to stems or lemmas
- Sentence segmentation

Tokenization Issues

Punctuation

- Separate most punctuation such as commas and periods that end phrases and sentences
- But keep the punctuation internal to words or tokens
  - Ph.D., m.p.h., AT&T, cap'n

Special characters occur in prices, dates, and numbers

- $45.55, 01/02/18, 65,500.005

Social media text has hashtags, URLs, and mentions

- #havingabadday, @john_doe, http://www.syr.edu

Clitics, such as "you're" and "don't"

- Choose whether to keep one token, separate into three tokens ("you," "'," "re") or two tokens ("do" and "n't"), or expand abbreviated words ("you" and "are")

Penn Treebank Tokenization

A commonly used tokenization scheme is the one used by Penn Treebank since many POS taggers and parsers are defined by that corpus.

It chooses to separate clitics into two tokens, keep together hyphenated words and numbers, but otherwise separate out punctuation.

- "The San Francisco-based restaurant", they said, "doesn't charge $10".
- Tokens: " The San Francisco-based restaurant " , they said , " does n't charge $ 10 " .

Capitalization Decision

Case folding is a type of normalization.

- For speech and information retrieval, everything is mapped to lower case.
- For sentiment analysis, information extraction, and machine translation, capitalization is quite helpful.
  - For example, in identifying proper nouns

Sentence Segmentation

School of Information Studies
Syracuse University

Importance of Punctuation

So far, we have discussed what words to keep and possible alternate forms of words (i.e., *word normalization*, through lowercasing, stemming, and lemmatization).

Note that, for further steps of language processing, we need to keep all the punctuation as tokens.

Punctuation determines the clauses of a sentence and can profoundly affect the meaning.

- From the book *Eats, Shoots and Leaves: The Zero Tolerance Approach to Punctuation,* by Lynne Truss, a collection of quotes: http://www.goodreads.com/work/quotes/854886-eats-shoots-leaves-the-zero-tolerance-approach-to-punctuation
- Another example seen on a t-shirt:

  Let's eat Grandma!
  Let's eat, Grandma!
  Commas save lives!

Sentence Segmentation

Punctuation not only shows the internal structure of sentences, but is crucial in determining the end of sentences.

EndOfSentence is determined by a lot of white space or punctuation !, ?, .

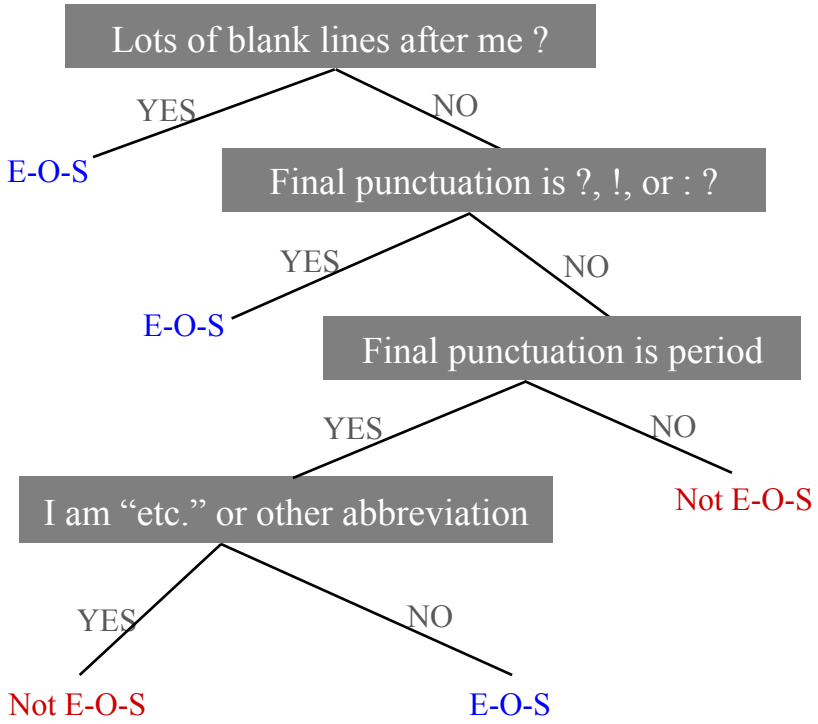
- !, ? are relatively unambiguous
- Period "." is quite ambiguous
  - Sentence boundary
  - Abbreviations like Inc. or Dr.
  - Numbers like .02% or 4.3

Treat this as a classification problem.

- Looks at a "." (or the word with the "." at the end)
- Decides EndOfSentence/NotEndOfSentence
- Classifiers: hand-written rules, regular expressions, or machine learning

Classify Whether a Word Is End-of-Sentence

An example of one way to classify is a decision tree.



Slides in this section are from Dan Jurafsky

Classification Problem Features

Each property used in the decision tree to decide which branch to take is called a feature of the word

Features for end-of-sentence decision

- Word with "." is on list of abbreviations
- Word shape features
  - Case of word with ".": upper, lower, cap, number
  - Look at word after "." to see if it begins a new sentence:
    - Case of word after ".": upper, lower, cap, number
- Numeric features
  - Length of word with "."
  - Probability(word with "." occurs at end of sentence)
  - Probability(word after "." occurs at beginning of sentence)