REGULAR EXPRESSIONS AND MORPHOLOGY

LU XIAO
LXIA004@SYR.EDU
213 HINDS HALL

ADOPTED SOME MATERIALS DEVELOPED IN PREVIOUS COURSES BY NANCY MCCRACKEN, LIZ LIDDY AND OTHERS; AND SOME INSTRUCTOR RESOURCES FOR THE BOOK "SPEECH AND LANGUAGE PROCESSING" BY DANIEL JURAFSKY AND JAMES H. MARTIN

THIS WEEK'S OFFICE HOUR

- Jenna
  - Wednesday: 2:00 – 4:00 pm.
  - Thursday: 2:00 – 4:00 pm.

- Lu
  - Thursday: 9:00 – 10:00 am.
  - Friday: 4:30 – 5:30 pm.

Shared Task For Chinese Grammatical Error Diagnosis

- https://sites.google.com/view/nlptea2018/shared-task
- "The goal of this shared task is to develop NLP techniques to automatically diagnose (including correction, at least partially) grammatical errors in Chinese sentences written by CFL learners. Such errors are defined as **redundant words** (denoted as a capital "**R**"), **missing words** ("**M**"), **word selection errors** ("**S**"), and **word ordering errors** ("**W**"). The input sentence may contain one or more such errors. The developed system should indicate which error types are embedded in the given sentence and the position at which they occur."

Example 1

Input: (sid=00038800481) 我根本不能了解这妇女辞职回家的现象。在这个时代，为什么放弃自己的工作，就回家当家庭主妇？

Output: 00038800481, 6, 7, S

00038800481, 8, 8, R

(Notes: "了解" should be "理解". In addition, "这" is a redundant word.)

Regular Expressions (RE) And Text Searching

- Regular expressions (a.k.a. regex, regexp or RE) are essentially a tiny, highly specialized programming language embedded inside other languages such as Python, Java, Perl, php, etc.

- Regular expressions are a compact textual representation of a set of strings – Pattern
  - "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

INTRODUCTION TO THE NOTATION OF RE

- Talk by Dan Jurafsky
- This introduction to RE is part of the publically available lectures from the Coursera course in Natural Language Processing with Dan Jurafsky and Chris Manning:

  https://class.coursera.org/nlp/lecture/index

MORE DETAILS ABOUT REGULAR EXPRESSIONS

Basic Regular Expression Notation

- Summary of basic notations to match single characters and sequences of characters

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)?/`	Zero or one; optional
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

In these slides, we use the (Perl) convention that regular expressions are surrounded by / - Python uses “”

Regular Expressions

- Anchors
  - Constrain the position(s) at which a pattern may match
  - Think of them as "extra" alphabet symbols, though they actually match ε (the zero-length string):
    - `/^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*

Difference between \b and \B in regex (stackoverflow thread)

Regular Expressions

- Escapes
  - A backslash "\" placed before a character is said to "escape" (or "quote") the character. Some situations of using escapes:
    1. **Meta-characters**: The characters which 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).

Regular Expressions

- Escapes (continued)
  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/backreferences**: "\1", "\2", etc.

Regular Expressions

- Greediness
  - Regular expression counters/quantifiers which allow for a regular language to match a variable number of times (i.e., the Kleene star, the Kleene 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":
      - `/\w+.*\w+/` matches the entire string, and
      - `/\w+.*?\w+/` matches just "From each"

Regular Expression Examples

Character classes and Kleene symbols

[A-Z] = one capital letter

[0-9] = one numerical digit

[st@!9] = s, t, @, ! or 9 (equivalent to using | on single characters)

[A-Z] matches G or W or E (a single capital letter)

does not match GW or FA or h or fun

[A-Z]+ = one or more consecutive capital letters

matches GW or FA or CRASH

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

[A-Z]* = zero, one or more consecutive capital letters

matches on EAT or I

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 (Cont'd)

[A-Za-z] = any single letter

so [A-Za-z]+

Will it match on these strings (separately)?

bi, weekly, bi-weekly, yes@SU, IBM

It matches on any word composed of only letters

a shortcut for [A-Za-z0-9_] is \w

so /\w+/ will match on Information, ZANY, rattskellar and jeuvbaew

/\s/ will match whitespace

So will /\w+\s\w+/ match the following strings (separately)?

real estate Gen Xers

Regular Expression Examples (Cont'd)

Some longer examples:

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

Will it match on these strings (separately)?

Intel c09yt745 IBM series5000

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

Will it match on these strings (separately)?

The dog died! he said, " The dog died! "

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

Will it match on these strings (separately)?

The dog died! "he said, " The dog died!"

(\w+ats?\s)+

parentheses define a pattern as a unit, so the above expression will match:

"Fat cats eat Bats that Splat "

False Negative (Type II) and False Positive (Type I)

- Find all the instances of the word "the" in a text.
  - **/the/**
    - But we will miss finding "The" in the text with this re
  - False negatives (Type II): not matching things that we should have matched (*The*)
  - **/[tT]he/**
    - Either t or T will be a match – [tT]
    - But we will falsely match words like there, Their, etc.
  - False positives (Type I): matching strings that we should not have matched (there, then, other)

How To Use Regular Expressions in Python

Option 1:

- 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

Option 2:

- Use **re.match("<regular expr>", string)** method

More 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 that match and returns them in a list
- 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.

More at Python docs: https://docs.python.org/3.4/library/re.html

Substitution With Regular Expressions

- 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 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.”

Helpful Regular Expression Websites

1. Free interactive testing/learning/exploration tools:
   a. Regular Expression tester:
      http://regexpal.com/

2. Tutorials:
   a. The Python Regular Expression HOWTO:
      http://docs.python.org/howto/regex.html
   or https://docs.python.org/3/howto/regex.html
      A good introduction to the topic, and assumes that you will be using Python.

3. Regular expression summary pages
   a. Dave Child's Regular Expression Cheat Sheet from addedbytes.com
      http://www.cheatography.com/davechild/cheat-sheets/regular-expressions/

BASIC TEXT PROCESSING: MORPHOLOGY WORD STEMMING

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
- To do these tasks, we need to use morphology

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
- **Morpheme**: Minimal Units Of Meaning. In other words, a morpheme is a minimal unit of meaning in a word
- We can usefully divide morphemes into two classes
  - Stems: The core meaning-bearing units
  - Affixes: Bits and pieces that adhere to stems to change their meanings and grammatical functions: prefixes, infixes, suffixes, circumfixes

Examples

un- + happy = *unhappy*

Prefix **Stem**

Prefixes appear in front of the stem to which they attach

blooming- + absolutely = *absobloominglutely*

Infix **Stem**

Infixes appear inside the stem to which they attach

emote + **-ion**

Stem **Suffix**

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

Spelling and sound changes often occur at the boundary of *fusional* languages, like English - Very important for NLP

English Morphology

- We can further divide morphology up into two broad classes
  - **Inflectional**
  - **Derivational**
- Inflectional morphology concerns the combination of stems and affixes where the resulting word:
  - Has the same word class (e.g., noun, verb, etc.) as the original
  - Serves a grammatical/semantic purpose that is different from the original but is nevertheless transparently related to the original
- *Examples:*
  - *apple* – noun; *apples* – still a noun — does not change the grammatical category (part of speech)
  - *apple* (singular), *apples* (plural) — mark the grammatical subclass to which it belongss
  - both *apple* and *apples* refer to the fruit — does not change the overall meaning

English has few inflections

Derivational Morphology

- 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

- Derivation is often limited to a certain group of words
  - You can **Clintonize** the government, but you can't **Bushize** the government
  - This restriction is partially phonological

Derivation In English

- English has many derivational affixes/suffixes, and they are regularly used to form new words
  - Part of this is cultural -- English speakers readily accept newly introduced terms

- Example: Verbs and Adjectives to Nouns

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

Inflection & 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

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

Creation Of New Words

- **Derivation** - adding prefixes or suffixes to form a new word
  - Clinton → Clintonize
- **Compounding** - combining two existing words
  - home + page → homepage
- **Clipping** - shortening a polysyllabic word
  - Internet → net
- **Acronyms** - take initial sounds or letters to form new word
  - Scuba → Self Contained Underwater Breathing Apparatus
- **Blending** - combine parts of two words
  - motor + hotel → motel
  - smoke + fog → smog
- **Backformation**
  - resurrection → resurrect

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
  - 2 genders (masculine and feminine) in Romance languages like French, Spanish, Italian
  - 3 genders (masc, fem, and neuter) in Germanic and Slavic languages
  - More are called noun classes – Bantu has up to 20 genders
  - Gender is sometimes explicitly marked on the word as a morpheme, but sometimes is just a property of the word

How Does NLP Make Use Of Morphology?

- Stemming
  - Strip prefixes and / or suffixes to find the base root, which may or may not be an actual word
    - Spelling corrections not required
- Lemmatization
  - Strip prefixes and / or suffixes to find the base root, which will always be an actual word
    - Spelling corrections are crucial
    - Often based on a word list, such as that available at WordNet
- Part of speech guessing
  - Knowledge of morphemes for a particular language can be a powerful aid in guessing the part of speech for an unknown term

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
  - M.F. Porter. An algorithm for suffix stripping. 1980, 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.

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
…			

From Dan Jurafsky

Some Other Stemmers For English

- Paice-Husk Stemmer
  - Simple iterative stemmer; rather heavy when used with standard rule set
- Krovetz Stemmer
  - Light stemmer; removes inflections only; removal of inflections is very accurate (actually a lemmatizer)
  - Often used as a first step before using another stemmer for increased compression
- Lovins Stemmer
  - Single-pass, context-sensitive, longest match stemmer; not widely used
- Dawson Stemmer
  - Complex linguistically targeted stemmer based on Lovins; not widely used

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

- But may use a lexicon to find actual words

Guessing The Part Of Speech

- English is continuously gaining new words on a daily basis
- And new words are a problem for many NLP systems
  - New words won't be found in the MRD or lexicon, if one is used
- How might morphology be used to help solve this problem?
- What part of speech are:
  - clemness
  - foramation
  - depickleated
  - outtakeable

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: 3rd person sing. 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'

BASIC TEXT PROCESSING: SENTENCE SEGMENTATION

Importance Of Punctuation

- So far we have discussed what words to keep and possible alternate forms of words, i.e. *word normalization*, through lower casing, 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!

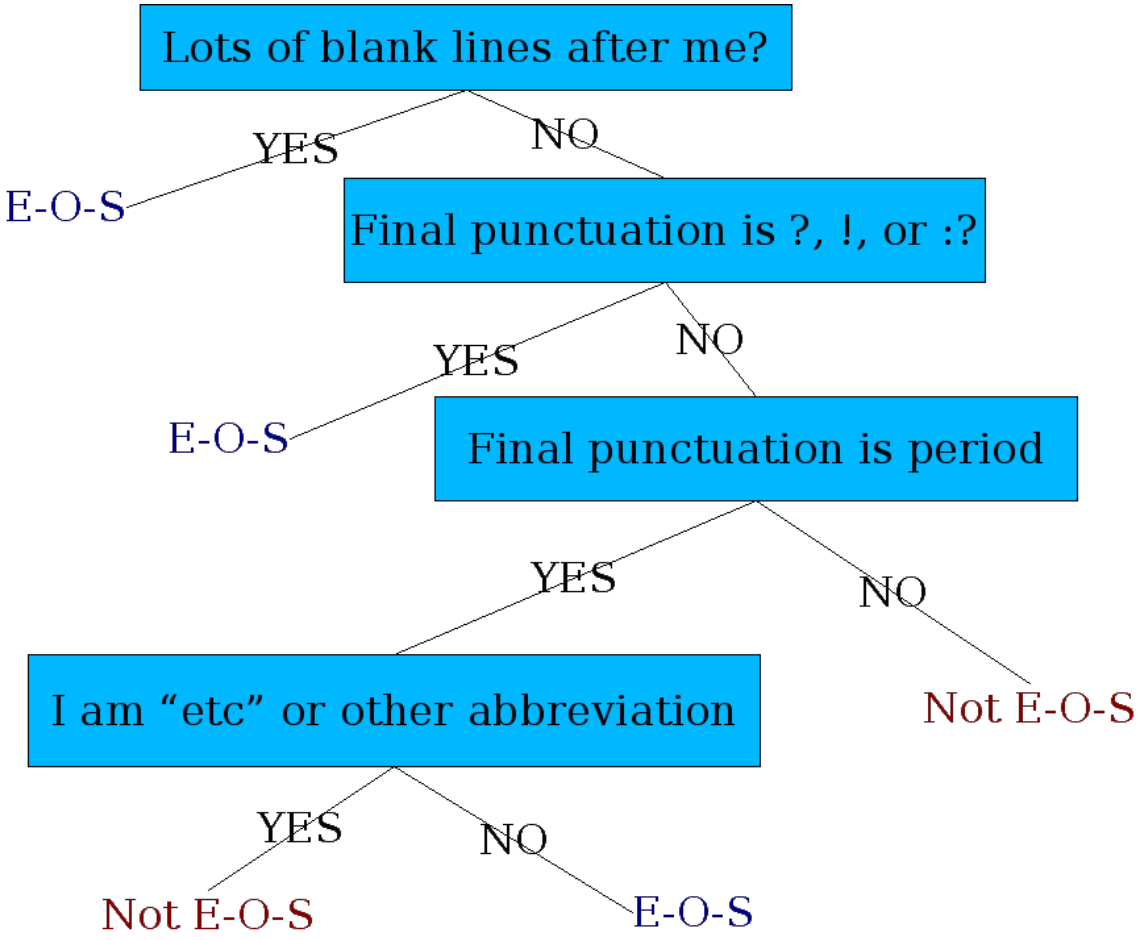
Commas save lives ☺

Sentence Segmentation

- Punctuation not only shows internal structure of sentences, but is crucial in determining the end of sentences.
- EndOfSentence determined by lots 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-s)
    - Probability(word after "." occurs at beginning-of-s)