Week 3:

Regular expression& Morphological analysis

School of Information Studies
Syracuse University

House Keeping notes

- Individual Assignment #1: available and will be due on **10/3**.
- Final project– Form your group before **10/3**, and sign up your names in the “Group”
- Mini-talk will start in next week

Overview

- Lecture:
  - Regular expression
  - Morphology
  - Stemming
- Lab

Regular Expressions: Basics

School of Information Studies
Syracuse University

Review

- Corpus statistics: static analysis-- description
  - Word frequency
  - N-gram frequency
  - Mutual information
- Language models/n-gram models-- prediction
  - probability of a sentence
  - probability of an upcoming word
- Lab: self-defined function, bigram,

Overview: Regular expressions

- Regular expressions (a.k.a. regex, or RE) are essentially a tiny, highly specialized programming language
  - embedded inside Python, Perl, Java, php and other languages
- Be used to specify the rules for a pattern to match any set of possible strings
- "Does this string match the pattern?", or "Is there a match for the pattern anywhere in this string?"

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, it is the pattern 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	Wegmans
[1234567890]	Any digit	Any 3 of them, iPhone 13

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	Fourth of July
[0-9]	Any digit	Any 3 of them, iPhone 9

Negation in Character Classes (not)

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 lower case letter	Chapter3:
[^Zz]	Not Z or z	Zzzzzzzozzz
`[^s^]`	Not s or ^	Base ^ exp
a^b	The pattern a^b	Look up a^b now

Note: If a ^ character does not occur in the first position inside a square bracket, then it just means the character itself

Disjunction (or)

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`	Either word regardless the capitalization of the first letter	A woodchuck is the same as a groundhog!

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

Repetition operators and "." (quantifiers)

Pattern	Matches	Example
colou?r	Optional previous char (0 or 1)	color colour
o*h!	0 or more of previous char (0:N)	h! oh! ooh! ooooh!
o+h!	1 or more of previous char (1:N)	oh! ooh! oooh! ooooh!
beg.n	"." matches any character	begin begun begun beg3n
Xyz{m}	m of previous char	abxXyzzz (m=3)
abc{m, n}	Between m and n of previous char	xycabcclby tfeabcccc (1,5)

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	`Syracuse University`
`^[^A-Za-z]`	Any char not a letter at the beginning	`1  "Hello"`
`\.$`	A dot at the end	`The end.`
`.$`	Any char at the end	`The end?` `The end`

Basic Notation Summary

- 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)?/	Question mark: 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

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, or PLAte

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
- `/\bour\b/` "Word" boundary
- `/\B23\B/` "Word" **non**-boundary
- *Note: **word** in RE is defined as any sequence of digits, letters, and underscores.*

Parentheses

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

Substitution

- Using one pattern to replace another
- s/ one pattern (being replaced)/ new pattern (replacement)
- e.g. s/colour/color

- **Capture group**: use of *parentheses* to store a matched pattern in memory
- Example: put all integers within angle brackets:

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

  - \1 is a **register,** referring to the first capture group– sub-match in the parentheses: [0-9]+

Escapes

- A backslash "\" placed before a character is said to "escape" (or "quote") the character. The commonly-used escapes include:
  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: []()\{}|^$.?+*\

  2. "Special" escapes (from the "C" language):

     newline:\n

     return: \r

     tab:\t

Escapes (cont)

3. **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_]`

4. **Registers**: `\1`, `\2`, etc.

Greediness

Regular expression quantifiers are inherently *greedy:*

- 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 (lazy),*** by placing a question mark(?) after it. Non-greedy quantifiers will at first match the **minimum** number of times.
  - Example #1:
    - greedy: /^.*t/ a076bt876xytdx
    - non-greedy: /^.*?t/ a076bt876xytdx
  - Example #2, against the string *Hello! This is Siri.*
    - `/\b\w+.*/` matches the entire string
    - `/\b\w+.*?/` matches just *Hello* as the first return
    - `/\b\w+?.*?/` how about this one?

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! "*

(\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* "

How to use Regex in Python

Option 1:

- the regex 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 the function *re.match*

  re.match(“<regular expr>”, string)

  which combines compile with the match function

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 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.”

Useful Resources

- Regular expressions 101: https://regex101.com/
- Regular expression cheat sheet: http://web.mit.edu/hackl/www/lab/turkshop/slides/regex-cheatsheet.pdf
- The Python Regular Expression HOWTO: https://docs.python.org/3/howto/regex.html

In-class activity

- Write a regex to find the following texts :
  - lower case alphabetic strings ending with *t*
    - *e.g. dogt*

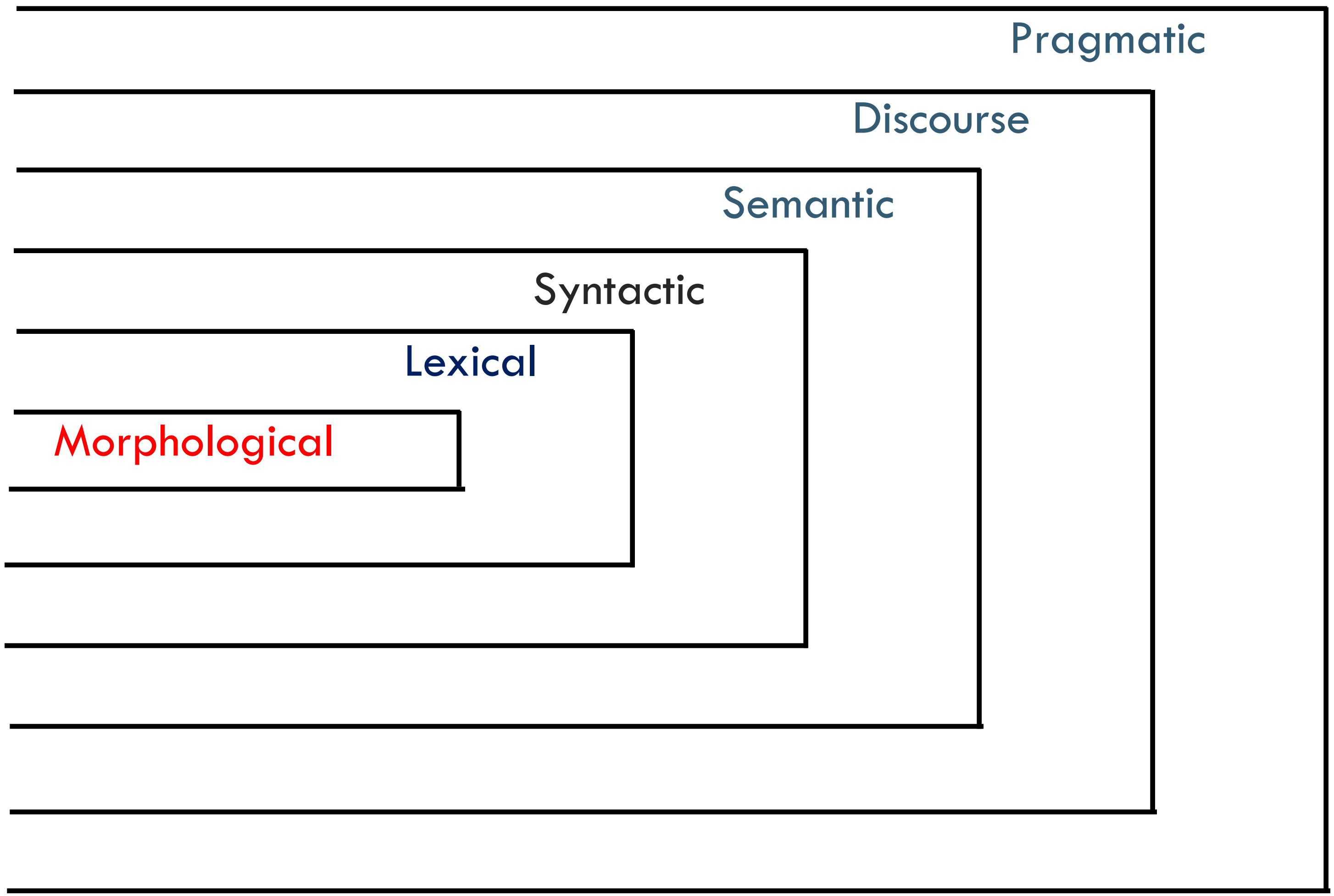
  - the set of all alphabetic strings with two consecutive repeated words (separated by space)
    - e.g. cat cat

  - all strings that start at the beginning of the text with an integer and that end at the end of the text with an alphabetic word (separated by space, any relevant punctuation, line breaks):
    - E.g. 989 dog bca3432t cat

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
- 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 or Chinese

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 free or bound forms
  - 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
- Circumfixes appear at both the beginning and end of stem
  - *en-, -en* in *enlighten*

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

-ation	computerize	computerization
-ee	appoint	appointee
-er	teach	teacher
-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
- English has few inflections

Ambiguous Affixes

- Some affixes are ambiguous:
  - -er
    - Derivational: employ –er
    - Inflectional: Comparative –er Adjective + -er
  - -s or –es
    - Inflectional: Plural apple, apples
    - Inflectional: 3rd person Eat , eats
  - -ing
    - Inflectional Progressive he is training to be a teacher
    - Derivational “act of” he attended the training
- As with all other ambiguity in language, this morphological ambiguity creates a problem for NLP

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 (1980)
  - 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.
- Other stemmers available for English like the Lancaster stemmer.

Examples of Porter Stemmer Rules

Step 1a

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

Step 1b

```
(*v*)ing → ø  walking   → walk

(*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
…
```

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

Lab

School of Information Studies
Syracuse University

Tasks

1. Reading texts from files
2. Stemming and Lemmatization
3. Removing special characters and stopwords