CS463 – Natural Language Processing

Basic Text Processing:

- Regular Expressions
- Text Normalization
- Word Tokenization
- Lemmatization and Stemming
- Sentence Segmentation and Decision Trees
- Minimum Edit Distance

Regular Expressions

- A formal language for specifying text strings.
- Formally, a regular expression is an algebraic notation for characterizing a set of strings.
- A regular expression search function will search through a **corpus**, returning all texts that match a **pattern**.
  - The simplest kind of regular expression is a sequence of simple characters.
  - For example:

RE	Example Patterns Matched
/woodchucks/	"interesting links to woodchucks and lemurs"
/a/	"Mary Ann stopped by Mona's"
/!/	"You've left the burglar behind again!" said Nori

Regular Expressions

- Regular expressions are **case sensitive**. This means that the pattern /woodchucks/ will not match the string "Woodchucks".
  - We can solve this by using **square braces** []
  - The string of characters inside the braces [] specifies a **disjunction** of characters to match.

RE	Match	Example Patterns
/[wW]oodchuck/	Woodchuck or woodchuck	"Woodchuck"
/[abc]/	'a', 'b', *or* 'c'	"In uomini, in soldati"
/[1234567890]/	any digit	"plenty of 7 to 5"

The use of the brackets [] to specify a disjunction of characters.

Regular Expressions: Disjunctions

- Use **dash -** inside brackets to specify any **one** character in a **range**.

Pattern	Matches	Example Patterns Matched
[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** can be applied using the **caret** ^ symbol
  - **Caret** means negation only when **first** in []

Pattern	Matches	Example Patterns Matched
`[^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* caret *b*	Look up a^b now

Regular Expressions: More Disjunction

- Woodchucks is another name for groundhog!
- The **pipe** | symbol for **disjunction**

Pattern	Matches
`groundhog|woodchuck`	groundhog woodchuck
`yours|mine`	yours mine
`a|b|c`	= [abc]
`[gG]roundhog|[Ww]oodchuck`	groundhog woodchuck Groundhog Woodchuck

Regular Expressions: ? * + .

- The question mark **(?)** Symbol means **zero** or **one** instance of the preceding character.
- The **Kleene** asterisk (***)** symbol means **zero** or **more** occurrences of the preceding character.
- The **Kleene** (+**)** symbol means **one** or **more** occurrences of the preceding character.
- The period (.) symbol is a **wildcard** expression that matches **any single** character it represents within the pattern (except a carriage return).

Pattern	Matches	
colou?r	Optional 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+	1 or more of previous char	baa baaa baaaa baaaaa
beg.n	Only 1 character	begin begun begun beg3n

Regular Expressions: Anchors ^ $

- **Anchors** are special characters that anchor regular expressions to particular places in a string.
- The caret (^) matches the **start of a line**.
  - The pattern /^The/ matches the word "The" only at the start of a line.
- The dollar sign $ matches the **end of a line**.
  - /^The dog\.$/ matches a line that contains only the phrase "The dog".

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

Regular Expressions: Boundary Anchors \b \B

- There are also two other anchors: \b matches a word boundary, and \B matches a non-boundary.
- For the purposes of a regular expression, a "word" is defined as any sequence of digits, underscores, or letters.
- Examples:
  - /\bthe\b/ matches the word "the" but not the word "other".
  - /\b99\b/ will match the string 99 in "There are 99 bottles of juice on the wall" (because 99 follows a space and precedes a space) but not 99 in "There are 299 bottles of juice 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).
- What will be the results of using the other anchor: \B in the previous examples knowing that it matches a non-word boundary?

Example:

- Suppose we wanted to write a RE to find cases of the English article "the". A simple (but incorrect) pattern might be:

/the/

- One problem is that this pattern will miss the word when it begins a sentence and hence is capitalized (i.e., The). This might lead us to the following pattern:

/[tT]he/

- But we will still incorrectly return texts with the embedded in other words (e.g., other or theology).
- So we need to specify that we want instances with a word boundary on both sides:

/\b[tT]he\b/

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
- 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

Text normalization

Text normalization

- Normalizing text means converting it to a more convenient, standard form.

1. **Tokenization -** Splitting a phrase, sentence, paragraph, or an entire text document into smaller units, such as individual words or terms.
2. **Lemmatization -** The task of determining that two words have the same root, despite their surface differences.
   - The words “sang”, “sung”, and “sings” are forms of the verb “sing”. The word sing is the common lemma of these words, and a lemmatizer maps from all of these to “sing”.
3. **Stemming -** We mainly just strip suffixes from the end of the word.
   - The words “caring”, “careful” are stemmed to “car”, and the words “history” and “historical” are stemmed to “histori”
4. **Sentence Segmentation -** We break up a text into individual sentences, using cues like periods or exclamation points.

Normalization

- Need to "normalize" terms
  - Information Retrieval: indexed text to 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***

Case folding

- Applications like IR (Information Retrieval): 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 (Machine Translate), Information extraction
  - Case is helpful (***US*** versus ***us*** is important)

Basic Text Processing

Word tokenization

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?

- A lemma is a set of lexical forms having
    - cat and cats = same lemma
- The wordform is the full inflected or derived form of the word.
    - 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?
  - 15 tokens (or 14)
  - 13 types (or 12)

How many words?

N = number of tokens

V = vocabulary = set of types

$|V|$ is the size of the vocabulary

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

Corpus	Tokens = N	Types = $|V|$
Shakespeare	884 thousand	31 thousand
Brown corpus	1 million	38 thousand
Switchboard telephone conversations	2.4 million	20 thousand
COCA	440 million	2 million
Google N-grams	1 trillion	13 million

Simple Tokenization in UNIX

- We can use command **tr** to tokenize the words by changing every sequence of non alphabetic characters to a newline (’A-Za-z’ means alphabetic, the -c option complements to non-alphabet, and the -s option squeezes all sequences into a Single character):

tr -sc 'A-Za-z’ ‘/n' < shakes.txt

The output of this command will be:

THE

SONNETS

by

William

Shakespeare

From

fairest

creatures

We

...

shakes.txt

THE SONNETS by William Shakespeare From fairest creatures We ….

Simple Tokenization in UNIX

- Now that there is one word per line, we can **sort** the lines, and pass them to **unique -c** which will collapse and count them:

```
tr -sc 'A-Za-z' '/n' < shakes2.txt | sort | uniq -c
```

with the following output:

1945 A
72 AARON
19 ABBESS
25 Aaron
6 Abate
1 Abates
5 Abbess
6 Abbey
3 Abbot
...

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. → ??

Basic Text Processing

Lemmatization and Stemming

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
- Machine translation
  - In Spanish: quiero ('I want'), quieres ('you want') same lemma as querer 'want'

Morphology

- It is the study of the internal structure of words.
- Morphology focuses on how the components within a word (stems, root words, prefixes, suffixes, etc.) are arranged or modified to create different meanings.
- Example: happy; un-happy; happy-ness; un-happy-ness
- **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

- 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

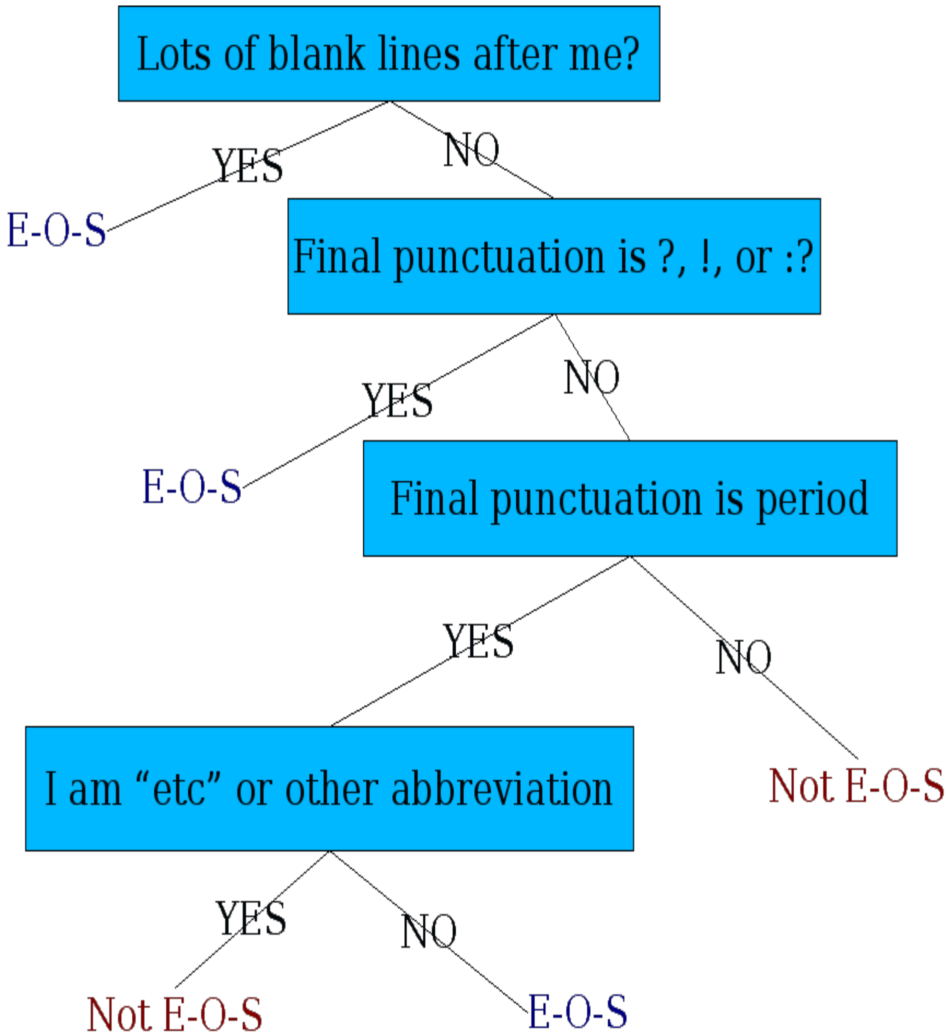
Basic Text Processing

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 (!).
- (?) and (!) are relatively unambiguous markers of sentence boundaries.
- (.) on the other hand, are more ambiguous.
  - Sentence boundary
  - Abbreviations like Inc. or Dr.
  - Numbers like .02% or 4.3
- Sentence tokenization methods work by building a binary classifier.
  - Look at a period "."
  - Decide EndOfSentence/NotEndOfSentence
  - Classifiers: hand-written rules, regular expressions, or machine-learning

Determining if a word is End-of-Sentence: Decision Tree



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.
- 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

Basic Text Processing

Minimum Edit Distance

How similar are two strings?

- Spell correction
  - The user typed "graffe"
  - Which is closest?
    - graf
    - graft
    - grail
    - giraffe

- Computational Biology
  - Align two sequences of nucleotides

    AGGCTATCACCTGACCTCCAGGCCGATGCCC
    TAGCTATCACGACCGCGGTCGATTTGCCCGAC

  - Resulting alignment:

    -AGGCTATCACCTGACCTCCAGGCCGA--TGCCC---
    TAG-CTATCAC--GACCGC--GGTCGATTTGCCCGAC

- Also for Machine Translation, Information Extraction, Speech Recognition

Minimum Edit Distance

- The minimum edit distance between two strings.

- It is the minimum number of editing operations.
  - Insertion
  - Deletion
  - Substitution

- Needed to transform one into the other.

Minimum Edit Distance

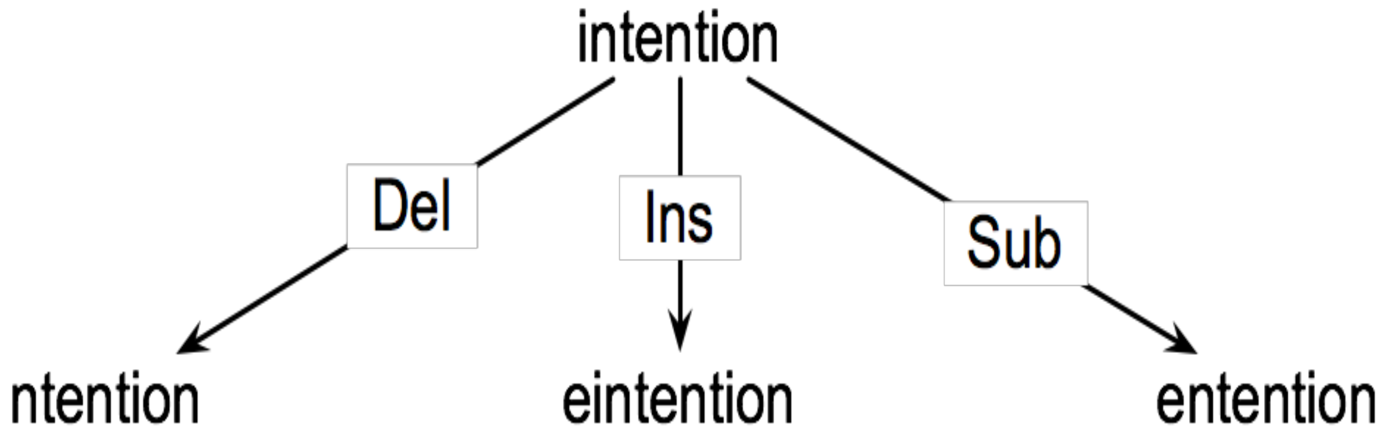
```
I N T E * N T I O N
| | | | | | | | | |
* E X E C U T I O N
d s s   i s
```

d-> delete
s-> substitution
i-> insert

- If each operation has cost of 1, then Distance between these is 5
- If substitution operation cost 2, then Distance between them is 8
  - The gap between **intention** and **execution**, for example, is 5 (delete an i, substitute e for n, substitute x for t, insert c, substitute u for n). 3 substitution + 1 insert + 1 delete =5

How to find the Min Edit Distance?

- Searching for a path (sequence of edits) from the start string to the final string:
  - **Initial state**: the word we're transforming
  - **Operators**: insert, delete, substitute
  - **Goal state**: the word we're trying to get to
  - **Path cost**: what we want to minimize: the number of edits



Defining Min Edit Distance

- For two strings
  - X of length n
  - Y of length m
- We define $D(i,j)$
  - the edit distance between $X[1..i]$ and $Y[1..j]$
    - i.e., the first i characters of X and the first j characters of Y
  - The edit distance between X and Y is thus $D(n,m)$

Minimum Edit Distance - Example

i n t e n t i o n
← *delete i*
n t e n t i o n
← *substitute n by e*
e t e n t i o n
← *substitute t by x*
e x e n t i o n
← *insert u*
e x e n u t i o n
← *substitute n by c*
e x e c u t i o n

Path from *intention* to *execution*.

$$D[i,j] = \min \begin{cases} D[i-1,j] + \text{del-cost}(source[i]) \\ D[i,j-1] + \text{ins-cost}(target[j]) \\ D[i-1,j-1] + \text{sub-cost}(source[i], target[j]) \end{cases}$$

$$D[i,j] = \min \begin{cases} D[i-1,j] + 1 \\ D[i,j-1] + 1 \\ D[i-1,j-1] + \begin{cases} 2; & \text{if } source[i] \neq target[j] \\ 0; & \text{if } source[i] = target[j] \end{cases} \end{cases}$$

Minimum Edit Distance - Example

Src\Tar	#	e	x	e	c	u	t	i	o	n
#	0	1	2	3	4	5	6	7	8	9
i	1	2	3	4	5	6	7	6	7	8
n	2	3	4	5	6	7	8	7	8	7
t	3	4	5	6	7	8	7	8	9	8
e	4	3	4	5	6	7	8	9	10	9
n	5	4	5	6	7	8	9	10	11	10
t	6	5	6	7	8	9	8	9	10	11
i	7	6	7	8	9	10	9	8	9	10
o	8	7	8	9	10	11	10	9	8	9
n	9	8	9	10	11	12	11	10	9	8