

# COMPUTATIONAL LINGUISTICS

**Athanasios N. Karasimos**

***akarasimos@gmail.com***

BA in Linguistics | School of English Language and Literature  
National and Kapodistrian University of Athens

**Lecture 2 | Wed 7 Mar 2018**

# BASIC TEXT PROCESSING: REGULAR EXPRESSIONS, TEXT NORMALIZATION & EDIT DISTANCE

# LECTURE I RECAP

# SUMMARY

- Space Odyssey 2001: A good way to understand the concerns of speech and language processing.
- Speech and Language technology relies on formal models, or representations, of knowledge of language at the levels of phonology and phonetics, morphology, syntax, semantics, pragmatics and discourse.
- The foundations of speech and language technology lie in computer science, linguistics, mathematics, cognitive science and psychology.
- The critical connection between language and thought has placed speech and language processing technology at the center of debate over intelligent machines.
- Revolutionary applications of speech and language processing are currently in use around the world.

# TASK: AMBIGUITY

- A perhaps surprising fact about these categories of linguistic knowledge is that most tasks in speech and language processing can be viewed as resolving **ambiguity** at one of these levels.
- TASK IV:
  - *Find an ambiguous example for each linguistic level.*
- 1. Phonetics & Phonology
- 2. Morphology
- 3. Syntax
- 4. Semantics
- 5. Pragmatics
- 6. Discourse

# TEXT NORMALIZATION I

((Token|Lemmat)ization|Stemming))

# TEXT NORMALIZATION

- **Text normalization** is the process of transforming text into a single canonical form that it might not have had before.
- Normalizing text before storing or processing it allows for separation of concerns, since input is guaranteed to be consistent before operations are performed on it.
- Text normalization requires being aware of what type of text is to be normalized and how it is to be processed afterwards; there is no all-purpose normalization procedure.

# TOKENIZATION

- Most of what we are going to do with language relies on first separating out or tokenizing words from running tokenization text, the task of tokenization.
- English words are often separated from each other by whitespace, but whitespace is not always sufficient.
- New York and rock 'n' roll are sometimes treated as large words despite the fact that they contain spaces, while sometimes we'll need to separate I'm into the two words I and am



# LEMMATIZATION

- Another part of text normalization is **lemmatization**, the task of determining that two words have the same root, despite their surface differences.
- For example, 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.
- Lemmatization is essential for processing morphologically complex languages like Russian, Turkish, Greek, Finnish, Hungarian, etc.

# STEMMING & SENTENCE SEGMENTATION

- **Stemming** refers to a simpler version of lemmatization in which we mainly just strip suffixes from the end of the word.
  - walking, walks, walked
- Text normalization also includes **sentence segmentation**: breaking up a text into individual sentences, using cues like sentence segmentation periods or exclamation points.

# REGULAR EXPRESSIONS

# WHAT IS A REGULAR EXPRESSION?

- A **regular expression** (RE, regex or regexp) is, in theoretical computer science and computational linguistics, a sequence of characters that define a *search pattern*. Usually this pattern is then used by string searching algorithms for "find" or "find and replace" operations on strings.
- The concept arose in the 1950s when the American mathematician Stephen Cole **Kleene** formalized the description of a regular language.
- Regular expressions are used in search engines, search and replace dialogs of word processors and text editors, in text processing utilities and in lexical analysis. Many programming languages provide regex capabilities, built-in, or via libraries.

# REGULAR EXPRESSION I

- A formal language for specifying text strings
- How can we search for any of these?
  - python
  - pythons
  - Python
  - Pythons

Pattern	Matches
/python/	“interesting links to <u>pythons</u> and snakes”
/a/	“M <u>a</u> ry Ann stopped by Mona’s”
/!/	“You’ve left the burglar behind again!” said Nori

# REGULAR EXPRESSIONS: DISJUNCTIONS

- Letters inside square brackets []

Pattern	Matches
[pP]ython	python, Python
[1234567890]	Any digit

- Ranges [A-Z] : the brackets can be used with the dash (-) to specify range any one character in a range

Pattern	Explanation	Matches
[A-Z]	An upper case letter	<u>A</u> thanasios Karasimos
[a-z]	A lower case letter	Re <u>g</u> ular expression
[0-9]	A single digit	Chapter <u>1</u> : Introduction

# REGULAR EXPRESSIONS: NEGATION IN DISJUNCTION

- Negations – The caret symbol [^]
  - Carat means negation only when it appears in [ ].

Pattern	Matches	Example
[^A-Z]	NOT an upper case letter	<u>O</u> nly I can write it.
[^Ss]	Neither 's' NOR 'S'	<u>S</u> pace, the final frontier.
[e^]	Either 'e' or '^'	Look up <u>h</u> ere.
a^b	The pattern 'a^b'	Check the text to find the hidden <u>a</u> ^ <u>b</u> .
[^\.]	NOT a period	<u>W</u> hat is the meaning of backslash (\)?

# REGULAR EXPRESSIONS: MORE DISJUNCTION

- We need a new operator, the disjunction operator, also called the **pipe** symbol `|`.
- The pattern `/cat|dog/` matches either the string *cat* or the string *dog*.

Pattern	Matches
<code>/dog cat pet/</code>	dog or cat or pet
<code>/yours mine/</code>	yours or mine
<code>/a b c/</code>	a, b or c ( <code>=<code>[abc]</code></code> )
<code>/[dD]og [cC]at [pP]et/</code>	Dog, dog, Cat, cat, Pet or pet
<code>/ferry ies/</code>	?



# REGULAR EXPRESSIONS: KLEENE SYMBOLS

- The **Kleene star** (\*) means “zero or more occurrences of the immediately previous character or regular expression”.
  - the `/a*/` means “any string of zero or more as”.
- The **Kleene plus** (+) means “one or more of the previous character”.
  - the expression `/[0-9]+/` is the normal way to specify “a sequence of digits”.
- The **question mark** (?) means “the preceding character or nothing”.
  - The example `/behaviour?r/` covers both *behavior* and *behaviour*.
- One very important special character is the **period** (.), a **wildcard** expression that matches any single character.
  - The expression `/beg.n/` >> any character between beg and n, such as *begin*, *beg’n*, *begun*.

# REGULAR EXPRESSIONS: KLEENE SYMBOLS

Pattern	Matches	Example
/colou?r/	Optional previous char	color or colour
/oo*h!/	0 or more of previous char	oh! ooh! oohh! ooooh! oooooh!
/oo+h!/	1 or more of previous char	ooh! oohh! ooooh! oooooh!
/beg.n/	any char	begin, began, begon, begun, beg3n, beg>n

# REGULAR EXPRESSIONS: ANCHORS

- **Anchors**  are special characters that anchor regular expressions to particular places in a string. The most common anchors are the caret `^` and the dollar sign `$`.
- The  **caret**  `^` matches the start of a line. The pattern `^The/` matches the word `The` only at the start of a line. Thus, the caret `^` has three uses: to match the start of a line, to indicate a negation inside of square brackets, and just to mean a caret.
- The  **dollar sign**  `$` matches the end of a line. So the pattern `$` is a useful pattern for matching a space at the end of a line, and `^The dog\.$/` matches a line that contains only the phrase `The dog`.
- There are also two other anchors: `\b` matches a word boundary, and `\B` matches a non-boundary.

# REGULAR EXPRESSIONS: ANCHORS

Pattern	Matches	Example
<code>^[A-Z]</code>	Any upper case letter at the beginning of a new line	<u>P</u> lease, close the door.
<code>^[^a-zA-Z]</code>	Any non-alphabetical symbol at the beginning of a new line	“ <u>_</u> Hello”, he said.
<code>\.\$</code>	A period at the end of a line.	The end <u>.</u>
<code>.\$</code>	Any symbol at the end of a line.	The end! <u>!</u> The end? <u>?</u>

# REGULAR EXPRESSION: OPERATOR PRECEDENCE

- This idea that one operator may take precedence over another, requiring us to sometimes use parentheses to specify what we mean, is formalized by the **operator precedence hierarchy** for regular expressions. The following table gives the order operator precedence of RE operator precedence, from highest precedence to lowest precedence.
  - Parenthesis ()
  - Counters \* + ? {}
  - Sequences and anchors the ^ \$
  - Disjunction |

# REGULAR EXPRESSIONS: OPERATORS

RE	Expansion	Match	First Matches
<code>\d</code>	<code>[0-9]</code>	any digit	Party_of_5
<code>\D</code>	<code>[^0-9]</code>	any non-digit	Blue_moon
<code>\w</code>	<code>[a-zA-Z0-9_]</code>	any alphanumeric/underscore	Daiyu
<code>\W</code>	<code>[^\w]</code>	a non-alphanumeric	!!!
<code>\s</code>	<code>[\r\t\n\f]</code>	whitespace (space, tab)	
<code>\S</code>	<code>[^\s]</code>	Non-whitespace	in_Concord

# REGULAR EXPRESSIONS: OPERATORS

<b>RE</b>	<b>Match</b>
*	zero or more occurrences of the previous char or expression
+	one or more occurrences of the previous char or expression
?	exactly zero or one occurrence of the previous char or expression
{ <i>n</i> }	<i>n</i> occurrences of the previous char or expression
{ <i>n</i> , <i>m</i> }	from <i>n</i> to <i>m</i> occurrences of the previous char or expression
{ <i>n</i> , }	at least <i>n</i> occurrences of the previous char or expression

# REGULAR EXPRESSIONS: AN EXAMPLE

- /the/
- /[tT]he/
- $\wedge$ b[tT]he\b/
- / $\wedge$ [a-zA-Z][tT]he $\wedge$ [a-zA-Z]/
- /( $\wedge$ [a-zA-Z])[tT]he( $\wedge$ [a-zA-Z]|\$)/



# REGULAR EXPRESSIONS: 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)**

## REGULAR EXPRESSIONS: ERRORS

- 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).

# REGULAR EXPRESSIONS: EXERCISES

- 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, we use machine learning classifiers
  - But regular expressions are used as features in the classifiers
  - Can be very useful in capturing generalizations
- Lets have a hands-on session!

# ASSIGNMENT I

**Write regular expressions for the following examples:**

- (a.) the set of all alphabetic strings.
- (b.) the set of all lower case alphabetic strings ending in a *b*.
- (c.) the set of all strings from the alphabet *a, b* such that each *a* is immediately preceded by and immediately followed by a *b*;
- (d.) all strings that have both the word *hobbit* and the word *orc* in them (but not, e.g., words like *hobbits* that merely *contain* the word *hobbit* – in any order!);
- (e.) the set of all the names and surnames of your classmates (do not forget the whitespace).

# ASSIGNMENT 1

- **Your goal is to use any regular expression techniques that we've learned so far to extract the filename, method name and line number of line of the stack trace (the underline bold part – parentheses included; do not forget that the rest of the text is just for distraction).**
- Skip     dalvikvm( 1553): threadid=1: uncaught exception
- Skip     ( 1553): FATAL EXCEPTION: main
- Skip     ( 1553): java.lang.StringIndexOutOfBoundsException
- Capture             widge7.List.makeView(ListView.java:1727)
- Capture             widg3t.List.fillDown(ListView.java:652)
- Capture             wldget.List.fillFrom(ListView.java:709)

# ASSIGNMENT I

- Write two regexp that matches all the items in the first column (positive examples), but none of those in the second (negative examples). You can choose which set of columns you will exclude (2 out of 3).

Positive	Negative		Positive	Negative		Positive	Negative
<p>pit spot spate slap two respite</p>	<p>pt Pot peat part</p>		<p>rap them tapeth apth wrap/try sap tray 87ap9th apothecary</p>	<p>aleht happy them tarpth Apt peth tarreth</p>		<p>daffgkking rafgkahe bafghk baffgkit daffgkking rafgkahe bafghk</p>	<p>fgok a fgk affgm dafffhk fgok afg.K aff gm</p>

# ASSIGNMENT I

**Give two or three examples from the given regexp OR describe what kind of matches you will get (i.e. the regexp `/d[a-k]{2,4}/` will give `daa`, `dbak`, `dfelk` etc. OR ‘words’ that start with `d` followed by two, three or four characters ranged from `a` to `k`)**

- (a.) `/[Tt]han(ks|x)/`
- (b.) `/\d\s\w\d\s\w\sAlphanumeric\sSequences?/`
- (c.) `/(b+eah)*/`
- (d.) `/(Jan|Feb|Mar|Apr|Mai|Jun|Jul|Aug|Sep|Oct|Noe|Dec)\s\d{4}/`
- (e.) `/^[Tt]he\s[a-z]+\./`

# TEXT NORMALIZATION II

Introduction and  
Examples



# TEXT NORMALIZATION

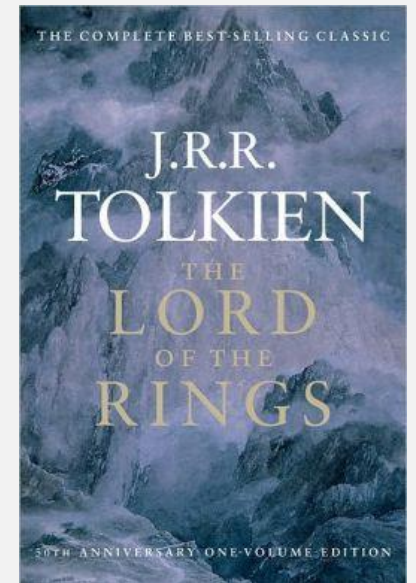
- Every NLP task needs to do text normalization:
  1. Segmenting/tokenizing words in running text
  2. Normalizing word formats
  3. Segmenting sentences in running text

# HOW MANY WORDS?

- I do uh main- mainly business data processing
  - Fragments, filled pauses
- 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?

- Three Rings for the Elven-kings under the sky,  
Seven for the Dwarf-lords in halls of stone,  
Nine for Mortal Men, doomed to die,  
One for the Dark Lord on his dark throne  
In the Land of Mordor where the Shadows lie.
- One Ring to rule them all,  
One Ring to find them,  
One Ring to bring them  
all and in the darkness bind them.
- **Type:** an element of the vocabulary.
- **Token:** an instance of that type in running text.
- How many?
  - Tokens:
  - Types:



# HOW MANY WORDS?

$N$  = number of tokens

$V$  = vocabulary = set of types

$|V|$  is the size of the vocabulary

	<b>Tokens = <math>N</math></b>	<b>Types = <math> V </math></b>
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. > ??

# TOKENIZATION: LANGUAGE ISSUES

- French
  - **L'ensemble** → one token or two?
    - **L ? L' ? Le ?**
    - Want **l'ensemble** to match with **un ensemble**
- German noun compounds are not segmented
  - **Lebensversicherungsgesellschaftsangestellter**
  - 'life insurance company employee'
  - German information retrieval needs **compound splitter**
- Greek
  - Τ' ανήλιαγα μέρη (τ'?)
  - Στου Μανώλη την ταβέρνα (στού?)

# 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

# WORD TOKENIZATION

- 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 (MaxMatch or Greedy) - The algorithm requires a dictionary (wordlist) of the maximum matching language.



# WORD TOKENIZATION: MAXMATCH

- Given a wordlist of Chinese, 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
  3. Move the pointer over the word in string
  4. Go to 2 (looping)

# WORD TOKENIZATION: MAXMATCH

- Input:               wecanonlyseeashortdistanceahead
- Output:             we canon l y see ash ort distance ahead

Doesn't generally work in English!

The algorithm works better in Chinese than English, because Chinese has much shorter words than English. We can quantify how well a segmenter works using a metric called **word error rate**.

We compare our output segmentation with a perfect hand-segmented ('gold') sentence, seeing how many words differ. The word error rate is then the normalized minimum edit distance in words between our output and the gold: the number of word insertions, deletions, and substitutions divided by the length of the gold sentence in words.

Modern probabilistic segmentation algorithms are even better.

# WORD NORMALIZATION

## and Stemming

# NORMALIZATION

- Need to “normalize” terms
  - Information Retrieval: indexed text & 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**
- Potentially more powerful, but less efficient

# CASE FOLDING

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

# LEMMATIZATION

- **Lemmatization** is the task of determining that two words have the same root, despite their surface differences.
- 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
- **TASK II:** *Present some problematic cases of lemmatization.*

# LEMMATIZATION AND MORPHOLOGY

- The most sophisticated methods for lemmatization involve complete morphological parsing of the word.
- **Morphology** is the study of the way words are built up from smaller meaning-bearing units called **morphemes**.
  - **Stems**: the core meaning-bearing units
  - **Affixes**: adding “additional” meanings of various kinds
    - 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**.
- One of the most widely used stemming Porter stemmer algorithms is the simple and efficient Porter (1980) algorithm.
- 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.  
*produces the following stemmed 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 STEMMER

## Step 1a

- sses > ss               caresses >caress
- ies > i                 ponies > poni
- ss > ss               caress > caress
- s > ∅                 cats > cat

## Step 1b

- (\*v\*)ing > ∅           walking > walk  
                                  sing > sing
- (\*v\*)ed > ∅           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 > ∅                 revival > reviv
- able > ∅              adjustable > adjust
- ate > ∅               activate > activ
- ...

# PORTER STEMMER

- *But what happens with the morphologically-rich languages?*
- 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** `p l pl` + **dan** `abl`
    - + **mis** `past` + **siniz** `2pl` + **casina** `as if`

# STEMMING

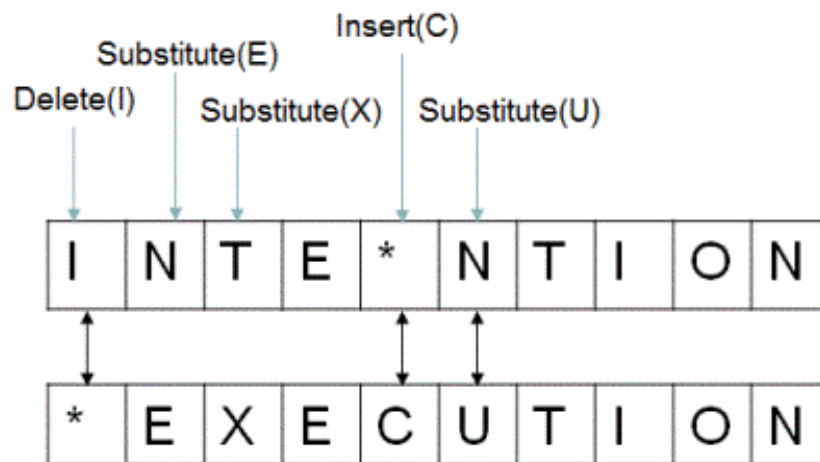
## Task III:

- Present four examples (English and/or Greek) that can be problematic for stemming.

## Task IV:

- Guess three possible errors of commission or omission.

<b>Errors of Commission</b>		<b>Errors of Omission</b>	
organization	organ	European	Europe
doing	doe	analysis	analyzes
numerical	numerous	noise	noisy
policy	police	sparse	sparsity



# MINIMUM EDIT DISTANCE

# MINIMUM EDIT DISTANCE: DEFINITION

- **Edit distance** gives us a way to quantify both of these intuitions (i.e. giraffe – graffe) about string similarity.
- The **minimum edit distance** between two strings is defined as the minimum number of editing operations (operations like insertion, deletion, substitution) needed to transform one string into another.
- It's much easier to see alignment this by looking at the most important visualization for string distances, an alignment between the two strings.

## MINIMUM EDIT DISTANCE: DEFINITION

- Given two sequences, an alignment is a correspondence between substrings of the two sequences. Thus, we say **I** aligns with the empty string, **N** with **E**, and so on. Beneath the aligned strings is another representation; a series of symbols expressing an operation list for converting the top string into the bottom string: **d** for deletion, **s** for substitution, **i** for insertion.

I	N	T	E	*	N	T	I	O	N
*	E	X	E	C	U	T	I	O	N

# MINIMUM EDIT DISTANCE

- If each operation has cost of 1
  - Distance between these is 5
- If substitutions cost 2 (Levenshtein)
  - Distance between them is 8

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

# USES OF EDIT DISTANCE IN NLP

- Evaluating Machine Translation and speech recognition

**R** Spokesman confirms senior government adviser was shot

**H** Spokesman said the senior adviser was shot dead

S I D I

- Named Entity Extraction and Entity Coreference
  - **IBM Inc.** announced today
  - **IBM** profits
  - **Stanford President John Hennessy** announced yesterday
  - for **Stanford University President John Hennessy**



# MINIMUM EDIT DISTANCE: HOW?

- 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

```
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
```

# MINIMUM EDIT DISTANCE

- **Task IV:**
  - Present the path from *decalcify* to *respecify*.
  - Present the path from *antennal* to *antihill*.
- **Task V:**
  - Give the minimum edit distance cost (both versions) of both paths.

# DEFINING MINIMUM 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  $D(n,m)$

# DEFINING MINIMUM EDIT DISTANCE (LEVENSHTEIN)

- Initialization

$$D(i, 0) = i$$

$$D(0, j) = j$$

- Recurrence Relation:

For each  $i = 1 \dots M$

For each  $j = 1 \dots N$

$$D(i, j) = \min \begin{cases} D(i-1, j) + 1 \\ D(i, j-1) + 1 \\ D(i-1, j-1) + \begin{cases} 2; & \text{if } X(i) \neq Y(j) \\ 0; & \text{if } X(i) = Y(j) \end{cases} \end{cases}$$

- Termination:

$D(N, M)$  is distance


# THE EDIT DISTANCE TABLE

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1									
#	0	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

# THE EDIT DISTANCE TABLE

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1									
#	0	1	2	3	4	5	6	7	8	9
#	E	X	E	C	U	T	I	O	N	

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 \\ D(i,j-1) + 1 \\ D(i-1,j-1) + \begin{cases} 2; & \text{if } S_1(i) \neq S_2(j) \\ 0; & \text{if } S_1(i) = S_2(j) \end{cases} \end{cases}$$



# THE EDIT DISTANCE TABLE

N	9	8	9	10	11	12	11	10	9	<b>8</b>
O	8	7	8	9	10	11	10	9	<b>8</b>	9
I	7	6	7	8	9	10	9	<b>8</b>	9	10
T	6	5	6	7	8	9	<b>8</b>	9	10	11
N	5	4	5	6	7	<b>8</b>	9	10	11	10
E	4	3	4	<b>5</b>	<b>6</b>	7	8	9	10	9
T	3	4	<b>5</b>	6	7	8	7	8	9	8
N	2	<b>3</b>	4	5	6	7	8	7	8	7
I	<b>1</b>	2	3	4	5	6	7	6	7	8
#	<b>0</b>	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

# COMPUTING ALIGNMENTS

- Edit distance isn't sufficient
  - We often need to **align** each character of the two strings to each other
- We do this by keeping a “**backtrace**”
- Every time we enter a cell, remember where we came from
- When we reach the end,
  - Trace back the path from the upper right corner to read off the alignment



# MINIMUM EDIT DISTANCE WITH BACKTRACE

<b>n</b>	9	↓ 8	↙↖↘ 9	↙↖↘ 10	↙↖↘ 11	↙↖↘ 12	↓ 11	↓ 10	↓ 9	↙ <b>8</b>	
<b>o</b>	8	↓ 7	↙↖↘ 8	↙↖↘ 9	↙↖↘ 10	↙↖↘ 11	↓ 10	↓ 9	↙ <b>8</b>	← 9	
<b>i</b>	7	↓ 6	↙↖↘ 7	↙↖↘ 8	↙↖↘ 9	↙↖↘ 10	↓ 9	↙ <b>8</b>	← 9	← 10	
<b>t</b>	6	↓ 5	↙↖↘ 6	↙↖↘ 7	↙↖↘ 8	↙↖↘ 9	↙ <b>8</b>	← 9	← 10	↖↙ 11	
<b>n</b>	5	↓ 4	↙↖↘ 5	↙↖↘ 6	↙↖↘ 7	↙↖↘ <b>8</b>	↙↖↘ 9	↙↖↘ 10	↙↖↘ 11	↖↙ 10	
<b>e</b>	4	↙ 3	← 4	↙↖ <b>5</b>	← <b>6</b>	← 7	↖↙ 8	↙↖↘ 9	↙↖↘ 10	↓ 9	
<b>t</b>	3	↙↖↘ 4	↙↖↘ <b>5</b>	↙↖↘ 6	↙↖↘ 7	↙↖↘ 8	↙ 7	↖↙ 8	↙↖↘ 9	↓ 8	
<b>n</b>	2	↙↖↘ <b>3</b>	↙↖↘ 4	↙↖↘ 5	↙↖↘ 6	↙↖↘ 7	↙↖↘ 8	↓ 7	↙↖↘ 8	↙ 7	
<b>i</b>	<b>1</b>	↙↖↘ 2	↙↖↘ 3	↙↖↘ 4	↙↖↘ 5	↙↖↘ 6	↙↖↘ 7	↙ 6	← 7	← 8	
<b>#</b>	<b>0</b>	1	2	3	4	5	6	7	8	9	
	<b>#</b>	<b>e</b>	<b>x</b>	<b>e</b>	<b>c</b>	<b>u</b>	<b>t</b>	<b>i</b>	<b>o</b>	<b>n</b>	

# COUNTING BACKTRACE COST

Base conditions:

$$D(i, 0) = i \qquad D(0, j) = j$$

Termination:

$$D(N, M) \text{ is distance}$$

Recurrence Relation:

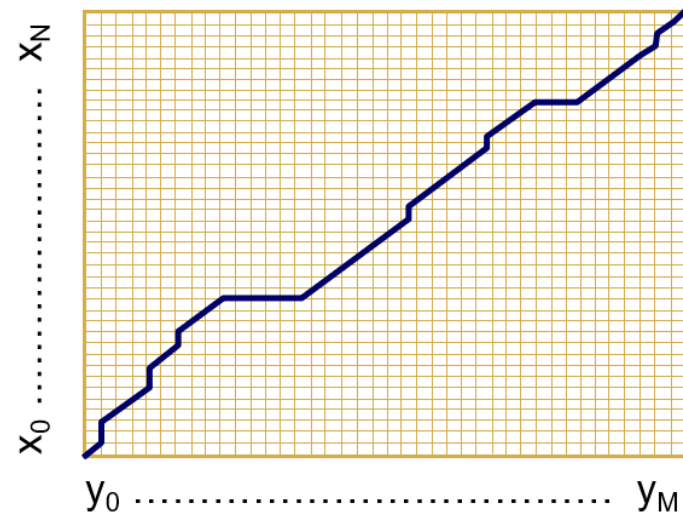
For each  $i = 1 \dots M$

For each  $j = 1 \dots N$

$$D(i, j) = \min \begin{cases} D(i-1, j) + 1 & \text{deletion} \\ D(i, j-1) + 1 & \text{insertion} \\ D(i-1, j-1) + \begin{cases} 2; & \text{if } X(i) \neq Y(j) \\ 0; & \text{if } X(i) = Y(j) \end{cases} & \text{substitution} \end{cases}$$

$$\text{ptr}(i, j) = \begin{cases} \text{LEFT} & \text{insertion} \\ \text{DOWN} & \text{deletion} \\ \text{DIAG} & \text{substitution} \end{cases}$$

# THE DISTANCE MATRIX



Every non-decreasing path  
from  $(0,0)$  to  $(M, N)$

corresponds to  
an alignment  
of the two sequences

An optimal alignment is composed  
of optimal subalignments

# MINIMUM EDIT DISTANCE TABLE

		i	n	t	e	r	e	s	t
	0	1	2	3	4	5	6	7	8
i	1	0	1	2	3	4	5	6	7
n	2	1	0	1	2	3	4	5	6
d	3	2	1	2	3	4	5	6	7
u	4	3	2	3	4	5	6	7	8
s	5	4	3	4	5	6	7	6	7
t	6	5	4	3	4	5	6	7	6
r	7	6	5	4	5	4	5	6	7
y	8	7	6	5	6	5	6	7	8

i n d u s t r e s t  
i n t e r e s t

# SUMMARY

# SUMMARY

- This lecture introduced a fundamental tool in language processing, the regular expression, and showed how to perform basic text normalization tasks including word segmentation and normalization, sentence segmentation, and stemming.
- We also introduce the important minimum edit distance algorithm for comparing strings.
  - The regular expression language is a powerful tool for pattern-matching.
  - Basic operations in regular expressions include concatenation of symbols, disjunction of symbols ( $\square$ ,  $|$ , and  $.$ ), counters ( $*$ ,  $+$ , and  $\{n,m\}$ ), anchors ( $^$ ,  $\$$ ) and precedence operators ( $(,)$ ).

# SUMMARY

- Word tokenization and normalization are generally done by cascades of simple regular expressions substitutions or finite automata.
- The Porter algorithm is a simple and efficient way to do stemming, stripping off affixes. It does not have high accuracy but may be useful for some tasks.
- The minimum edit distance between two strings is the minimum number of operations it takes to edit one into the other. Minimum edit distance can be computed by dynamic programming, which also results in an alignment of the two strings.
- NEXT TIME: *Language Modelling with N-Grams*

# ASSIGNMENT #1+#2

- 3-Tasks Assignments
  - One with regular expressions
  - One with stemming
  - One with minimum edit distance
- Deadlines: Fri 20/10/2017 & Mon 30/10/2017
- Submission: via email ([akarasimos@gmail.com](mailto:akarasimos@gmail.com)).



# WEEK READINGS

- Jurafsky D. & J. Martin (2008). SPEECH and LANGUAGE PROCESSING  
An introduction to Natural Language Processing, Computational Linguistics and  
Speech Recognition (2nd Edition + 3rd Edition). CHAPTER 2.
- Porter, M.F. (2001). [Snowball:A language for stemming algorithms](#). (optional)