CE4045 CZ4045 SC4002
Natural Language Processing

Text Normalization and Edit Distance

Dr. Sun Aixin

Words and Corpora

- Words don't appear out of nowhere.
  - In NLP, we typically work on a particular corpus (or a few corpora), to develop a system before deployment.
- A **corpus** is a computer-readable collection of text (or speech).
  - A corpus is also called a text **dataset** or simply a dataset
  - Example corpus can be a collection of online reviews for mobile phones.
  - Another example is all articles published by Straits Times

- Let's take a sentence from a corpus

> Singapore is most beautiful city in Asia, 11th most beautiful in the world!

  - In this sentence, there 13 words if we don't count punctuation marks as words, 15 words if we count punctuation.
    - Whether to tread punctuation as word, is task dependent.
    - Punctuation is critical for finding boundaries of things (commas, periods, colons) or for identifying some aspects of meanings (question marks, exclamation marks, quotations marks)

Corpora

- There are many languages (Ethnologue catalog: 7097 languages for now)
  - NLP algorithms are most useful when they apply across many languages.
  - A large number of NLP algorithms are developed and tested on English.
- **Code switching**: speakers or writers to use **multiple** languages in the context of a single conversation or situation.
- Even all texts are in English, texts could be in different genre: news, medical reports, phone conversations.
- Text also reflects demographic characteristics: age, gender, socioeconomic class
- Language changes over time

Source: https://techcommunity.microsoft.com/t5/ai-customer-engineering-team/speech-recognition-for-singlish/ba-p/3580687

Corpora: datasheet

- When developing computational models for language processing from a corpus, it's important to consider
  - who produced the language, in what context, for what purpose.
- A **corpora datasheet** specifies properties of a dataset like:
  - **Motivation**: Why was the corpus collected, by whom, and who funded it?
  - **Situation**: When and in what situation was the text written/spoken? Was the language originally spoken conversation, edited text, social media communication, monologue vs. dialogue?
  - **Language variety**: What language (including dialect/region) was the corpus in?
  - **Speaker demographics**: What was, e.g., age or gender of the authors of the text?
  - **Collection process**: How big is the data? If it is a subsample how was it sampled? Was the data collected with consent? How was the data pre-processed, and what metadata is available?
  - **Annotation process**: What are the annotations, what are the demographics of the annotators, how were they trained, how was the data annotated?
  - **Distribution**: Are there copyright or other intellectual property restrictions?

Lemma, Wordform, Word type, and Word token

- A **lemma** is the base form of a set of words in general having the same stem, the same major part-of-speech, and the same word sense.
  - Lemma cat: cat, cats.
  - Lemma break: breaks, broke, broken, and breaking
- The **wordform** is the full inflected or derived form of the word.
  - **Inflectional:** has the same word class as the original: cat → cats
  - **Derivational:** Changes of word class: care → careless, computer → computerize
  - For many tasks in English, wordforms are sufficient.
- Word **type** vs word **token**
  - **Types** are the number of distinct words in a corpus, or the size of the vocabulary.
  - **Tokens** refer to the occurrences of the words

Types: 10
Tokens: 13

Singapore is most beautiful city in Asia, 11th most beautiful in the world!

(not counting punctuations)

Text Normalization

- Tokenizing (segmenting) words
  - The task of segmenting running text into words (and/or symbols)
  - Pretty much a prerequisite to doing anything interesting
  - Also called word segmentation, word tokenization, and the tool we use is often called **tokenizer**
- Normalizing word forms
- Segmenting sentences

Tokenization

- Words in English are separated by white-spaces.
  - Mr. Sherwood said reaction to Sea Containers' proposal has been "very positive." In New York Stock Exchange composite trading yesterday, Sea Containers closed at $62.625, up 62.5 cents.
  - "I said, 'what're you? Crazy?' " said Sadowsky. "I can't afford to do that."

- White-space splitter gives words like:
  - cents.
  - said,
  - positive."
  - Crazy?'

Tokenization: more cases

- Word-internal punctuation
  - M.P.H. Ph.D. AT&T Google.com Yahoo!
- Other Examples:
  - Finland's capital → Finland? Finlands? Finland's?
  - Hewlett-Packard → Hewlett and Packard as two tokens?
  - state-of-the-art: break up hyphenated sequence.
  - co-education, lowercase, lower-case, lower case?
- How about names?
  - San Francisco: one token or two?
  - How do you decide it is one token? Then how about "San Francisco Airport"?
- What about numbers?
  - 3/20/91 Mar. 12, 1991 20/3/91
  - 55 B.C., B-52, This is my number: (800) 234-2333

Tokenization: Implementation

- Tokenization needs to be run before any other language processing,
  - It needs to be very fast.
  - Standard method for tokenization is to use deterministic algorithms based on regular expressions.
  - Example Python-based Natural Language Toolkit (NLTK)

```
>>> text = 'That U.S.A. poster-print costs $12.40...'
>>> pattern = r'''(?x)    # set flag to allow verbose regexps
...     ([A-Z]\.)+        # abbreviations, e.g. U.S.A.
...   | \w+(-\w+)*        # words with optional internal hyphens
...   | \$?\d+(\.\d+)?%?  # currency and percentages, e.g. $12.40, 82%
...   | \.\.\.            # ellipsis
...   | [][.,;"'?():-_`]  # these are separate tokens; includes ], [
... '''
>>> nltk.regexp_tokenize(text, pattern)
['That', 'U.S.A.', 'poster-print', 'costs', '$12.40', '...']
```

Example segmentation results

TreebankWordTokenizer

1. `` | The | San | Francisco-based | restaurant | , | '' | they | said | , | `` | does | n't | charge | $ | 10 | '' | .

WordPunctTokenizer

1. " | The | San | Francisco | - | based | restaurant | ," | they | said | , | " | doesn | ' | t | charge | $ | 10 | ".

PunktWordTokenizer

1. " | The | San | Francisco-based | restaurant | , | " | they | said | , | " | doesn | 't | charge | $10 | " | .

WhitespaceTokenizer

1. "The | San | Francisco-based | restaurant," | they | said, | "doesn't | charge | $10".

pattern

1. " | The | San | Francisco-based | restaurant | , | " | they | said | , | " | does | n't | charge | $ | 10 | " | .

TreebankWordTokenizer

1. `` | No | , | Dr. | David | worked | at | Yahoo | !
2. in | New | York | before | joining | Google.com | "

WordPunctTokenizer

1. " | No | , | Dr | . | David | worked | at | Yahoo | !
2. in | New | York | before | joining | Google | . | com | "

PunktWordTokenizer

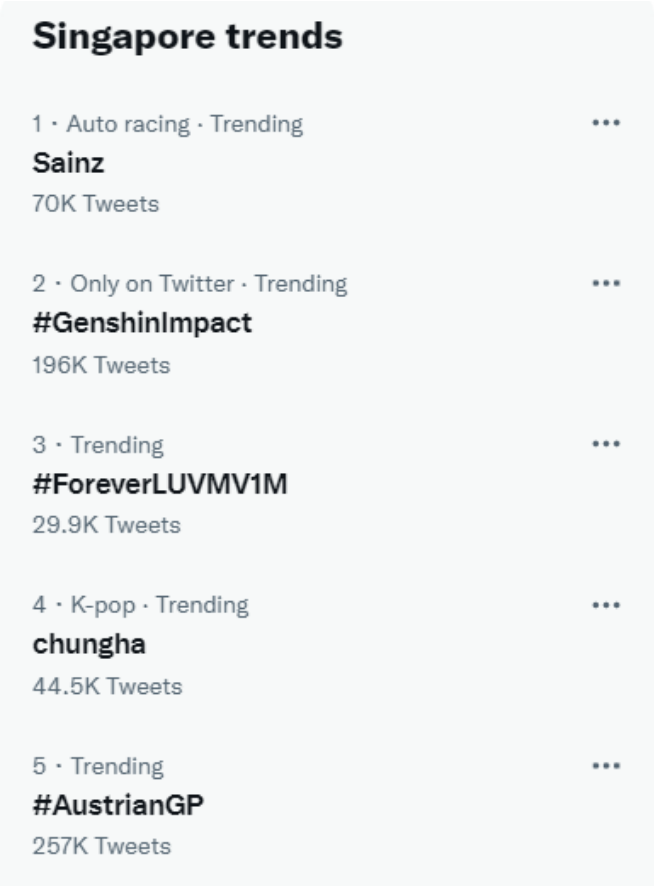
1. " | No | , | Dr. | David | worked | at | Yahoo | !
2. in | New | York | before | joining | Google.com | "

WhitespaceTokenizer

1. "No, | Dr. | David | worked | at | Yahoo!
2. in | New | York | before | joining | Google.com"

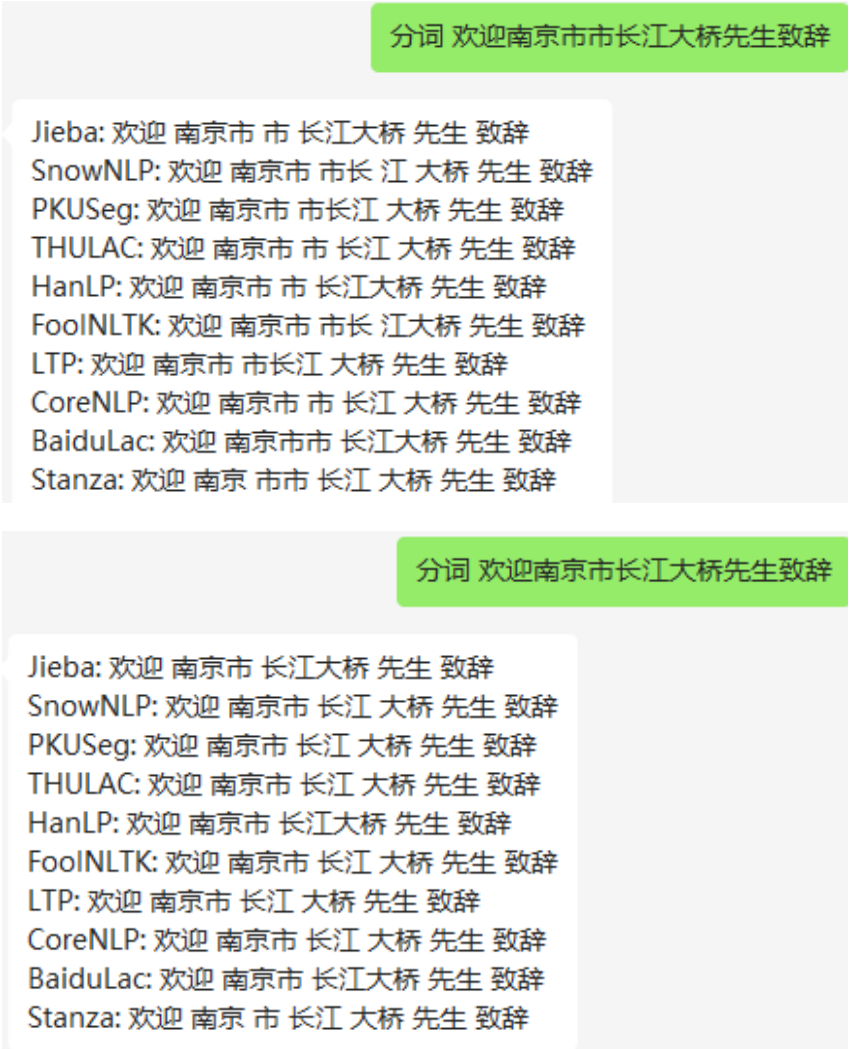
Specifical cases: white-spaces do not work here

- URL segmentation
  - www.dietsthatwork.com
  - www.choosespain.com
- Hashtag segmentation
  - #unitedbrokemyguitar
  - #manchesterunited
  - Hashtags allow Twitter users to track what many people (especially people whom you aren't already following) are reporting or thinking about a particular topic or event.



Tokenization is language dependent

- Chinese words composed of characters
  - Average word is 2.4 characters long.
- A simple segmentation algorithm based on dictionary: Maximum Matching or Maxmatch
  - Given a lexicon of Chinese, and a string
  - Start a pointer at the beginning of the string
  - Find the longest word in dictionary that matches the string starting at pointer
  - Move the pointer over the word in string
  - Go to 2
- State-of-the-art solutions are mostly probabilistic and deep learning
  - http://nlpprogress.com/chinese/chinese_word_segmentation.html
  - https://github.com/topics/chinese-text-segmentation



Tokenization: language issues

- German noun compounds are not segmented
  - **Lebensversicherungsgesellschaftsangestellter**
  - 'life insurance company employee'

- Japanese is more complicated, with multiple alphabets intermingled

  フォーチュン500社は情報不足のため時間あた$500K(約6,000万円)

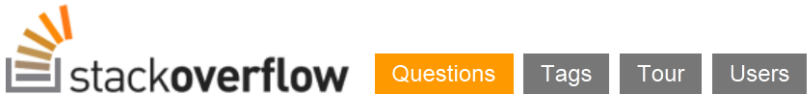
  Katakana Hiragana Kanji Romaji

- Arabic (or Hebrew) is basically written right to left, but with certain items like numbers written left to right

  استقلت الجزائر في سنة 1962 بعد 132 عاما من الاحتلال الفرنسي.

  - 'Algeria achieved its independence in 1962 after 132 years of French occupation.'

Tokenization is language dependent



Stack Overflow is a question and answer site for professional and enthusiast programmers. It's 100% free, no registration required.

EF6 - Value cannot be null. Parameter name: extent

0

I'm using EF6 code first to create my db. Everything was working well last night, now when i run update-database command, I get the following exception:

```
PM> update-database
Specify the '-Verbose' flag to view the SQL statements being appli
System.ArgumentNullException: Value cannot be null.
Parameter name: extent
   at System.Data.Entity.Utilities.Check.NotNull[T](T value, Strin
   at System.Data.Entity.Core.Mapping.StorageEntitySetMapping..cto
   at System.Data.Entity.ModelConfiguration.Edm.DbDatabaseMappingE
   at System.Data.Entity.ModelConfiguration.Edm.Services.TableMapp
   at System.Data.Entity.ModelConfiguration.Edm.Services.DatabaseM
   at System.Data.Entity.ModelConfiguration.Edm.Services.DatabaseM
   at System.Data.Entity.DbModelBuilder.Build(DbProviderManifest p
   at System.Data.Entity.DbModelBuilder.Build(DbConnection provide
   at System.Data.Entity.Internal.LazyInternalContext.CreateModel(
   at System.Data.Entity.Internal.RetryLazy`2.GetValue(TInput inpu
   at System.Data.Entity.Internal.LazyInternalContext.InitializeCo
   at System.Data.Entity.Internal.LazyInternalContext.get_CodeFirs
```

searchcode

binarytree lang:Java

Run searchcode locally? Want to run your own version of searchcode

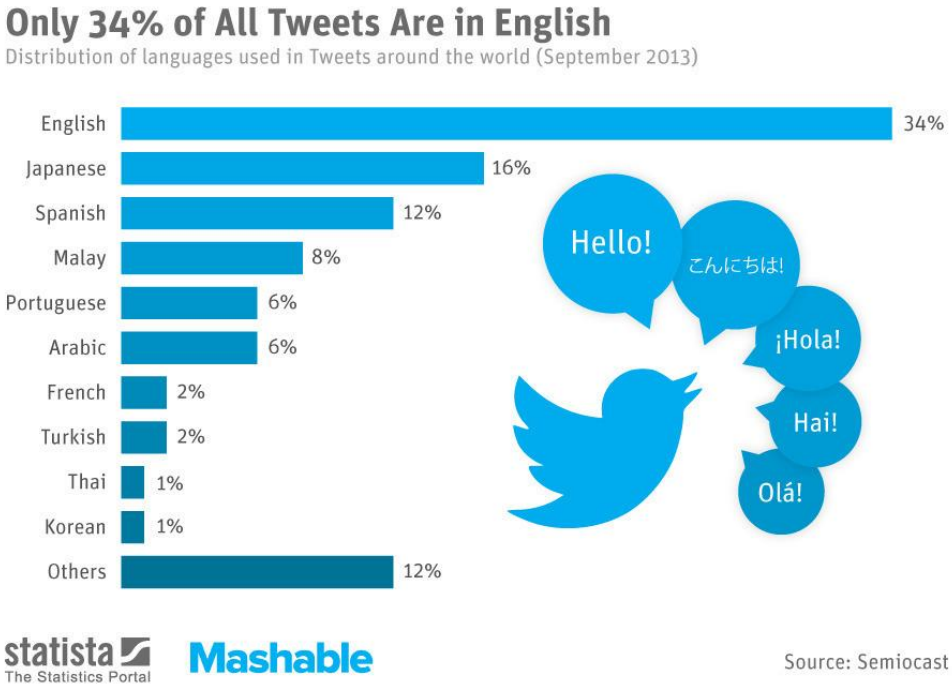
About 419 results

BinaryTree.java in rose_backup https://github.com/dpick/rose_backup.git | 171 lines | Java

```
1. // BinaryTree class; stores a binary tree.
2. //

8. // Also, the following tricky method:
9. // void merge( Object root, BinaryTree t1, BinaryTree t2 )
10. //                        --> Construct a new tree

14. /**
15.  * BinaryTree class that illustrates the calling of BinaryNode recursive
16.  * routines and merge.

17.  */
18. public class BinaryTree {
19.         /**
```

More than code switching

shd, ishould, shudd, shuld, shoud, shud, shld, sould, shouldd



http://languagetime.files.wordpress.com/2013/12/131217_twitter_sprachen_mashable_n.jpg

Word grouping by Brown Clustering

^01110111001 (43)	everything everythin everthing evrything everythang everythinq everythng eveything everythingg everyting evrythng errthang errything #everything errthing erything everythinggg errythang evrythin erthang 1thing erthing everythingggg jony everythig everytin everyhing everithing everythign everythn everyhting everythinqq everythink erythang everything-everythiing everythinggggg errrthang everythg everyword evreything everysong er'thing
^01110111010 (85)	nothing nothin nun nuthin nuttin nuffin 10x noting nthn nowt nuthn nuthing 100x nothingg nothn nothinq nutin notin nuffn nutn whatevers #nodisrespect nuttn nothinggg #dontmeantobrag 1000x zilch nothinn #nothing nothng nutting nufin nuin nout nothinqq nthng nthing nothinggg nufn nofin nothen nthin nottin nttn nought ntg nothinnn nothign n0thing nothig
^011101110110 (108)	something somethin sumthin sumthing sumn somthing sth sumthn sumtin smth somthin suttin sumin somethinq summin treaters somethingg someting sumfin smthn somethn somethng summat smthng smthing sum'n sumthng smthg somn sumtn smething sometin somethign sum10 soemthing somethinn somethinggg somethig sumpin somin sumting something/someone somtin somehting somthn someshit sumptin something- smtg thereabouts
^011101110111 (36)	anything anythin nething anythng anythingg anythinq anyting anythang anyth anytin anthing anythinggg anyfin vaart anythn anythign nethin nything anyhting #anything anything- anythg anythingggg nothing- anythink anythig anythinqq somethingggg nethng anyhing woodsen endsmeat anything/anyone aything anythiing enything

Source: http://www.cs.cmu.edu/~ark/TweetNLP/cluster_viewer.html

Text Normalization

- Tokenizing (segmenting) words
  - The task of segmenting running text into words (and/or symbols)
  - Pretty much a prerequisite to doing anything interesting
  - Also called word segmentation, word tokenization, and the tool we use is often called **tokenizer**
- Normalizing word forms
  - The task of putting words/tokens in a standard format, choosing a single normal form for words with multiple forms.
- Segmenting sentences

Text Normalization: case folding and lemmatization

- **Case folding**:
  - Mapping everything to lower case
    - "The" and "the" → "the"
    - "US" and "us" → "us"
    - "Fed" and "fed" → "fed"
  - Case folding is very helpful for generalization in many tasks, such as information retrieval or speech recognition.
  - Case folding is generally not done for information extraction, and machine translation.
- **Lemmatization**
  - The task of determining that two words have the same root, despite their surface differences.
    - "am", "are", and "is" have the shared lemma "be";
    - "dinner" and "dinners" both have the lemma dinner.
  - The most sophisticated methods for lemmatization involve complete morphological parsing of the word

Text Normalization: lemmatization

- **Morphology** is the study of the way words are built up from smaller meaning-bearing units called **morphemes**.
- There are two broad classes of morphemes
  - **stems**—the central morpheme of the word, supplying the main meaning
  - **affixes**—adding "additional" meanings of various kinds.
- Example:
  - word **fox** consists of one morpheme (the morpheme **fox**)
  - word **cats** consists of two: the morpheme **cat** and the morpheme **-s**.
- The most sophisticated methods for **lemmatization** involve complete morphological parsing of the word
  - A morphological parser takes a word like cats and parses it into the two morphemes cat and s.
  - Can be complex (and slow for some tasks)

Text Normalization: Stemming

- Stemming: a simpler but cruder method, which mainly consists of chopping off word-final stemming affixes.
  - Can be considered as a naive version of morphological analysis
  - The resultant stem of a word may not be a valid English word (or not really the stem)
  - There are many stemmers and the most widely used algorithm is Porter's stemmer

- **Porter's stemmer** does not need lexicon (or dictionary)
  - A set of rewrite rules that strip suffixes, run in series as a cascade
    - ING → ε (e.g., monitoring → monitor)
    - SSES→ SS (e.g., grasses → grass)
  - Example: Computerization
    - ization -> -ize computerize
    - ize -> ε computer

- Code: http://tartarus.org/martin/PorterStemmer/ (in many languages)

Porter Stemmer: Example

- Input (before stemming):

> 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.

- Output (after stemming):

> 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

- Try out: https://textanalysisonline.com/

Text Normalization

- Tokenizing (segmenting) words
  - The task of segmenting running text into words (and/or symbols)
  - Pretty much a prerequisite to doing anything interesting
  - Also called word segmentation, word tokenization, and the tool we use is often called **tokenizer**
- Normalizing word forms
  - The task of putting words/tokens in a standard format, choosing a single normal form for words with multiple forms.
- Segmenting sentences
  - Split a paragraph or document into sentences
  - And sometimes more difficult: https://www.youtube.com/watch?v=-c4CNB80SRc

Sentence segmentation

- Punctuation (like periods, question marks, and exclamation points) are the most useful cues for segmenting a text into sentences.
  - Question marks and exclamation points are relatively unambiguous markers of sentence boundaries, but not always true like Yahoo!
  - Periods are more ambiguous, e.g., Mr. or U.S.A. For this reason, sentence tokenization and word tokenization may be addressed jointly.

- In the Stanford CoreNLP toolkit, sentence splitting is rule-based, a deterministic consequence of tokenization;
  - A sentence ends when a sentence-ending punctuation (., !, or ?) is not already grouped with other characters into a token (such as for an abbreviation or number), optionally followed by additional final quotes or brackets.

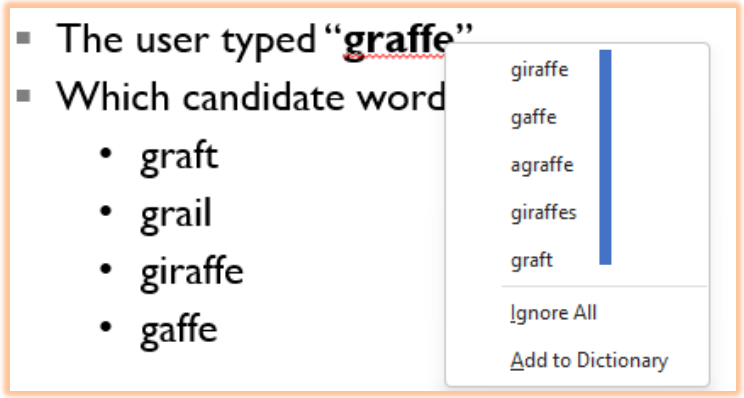
- Sentence segmentation/split/tokenization can be done by machine learning models.

Edit Distance

Dr. Sun Aixin

Measuring string similarity

- Much of natural language processing is concerned with **measuring how similar two strings are**.
- Example: Spelling correction
  - The user typed "**graffe**"
  - Which candidate word to choose?
    - graft
    - grail
    - giraffe
    - Gaffe



- Example: Coreference, the task of deciding whether two strings refer to the same entity: "IBM Inc. announced today" vs "IBM profits"
- Example: Evaluating machine translation and speech recognition
  - Spokesman confirms senior government adviser was shot
  - Spokesman said the senior adviser was shot dead

Minimum Edit Distance

- The **minimum edit distance** between two strings is the minimum number of editing operations needed to transform one into the other
  - Insertion
  - Deletion
  - Substitution
- Editing operations can be on letters or words depending on the application.
- Example: transforming "intention" to "execution" takes 5 operations

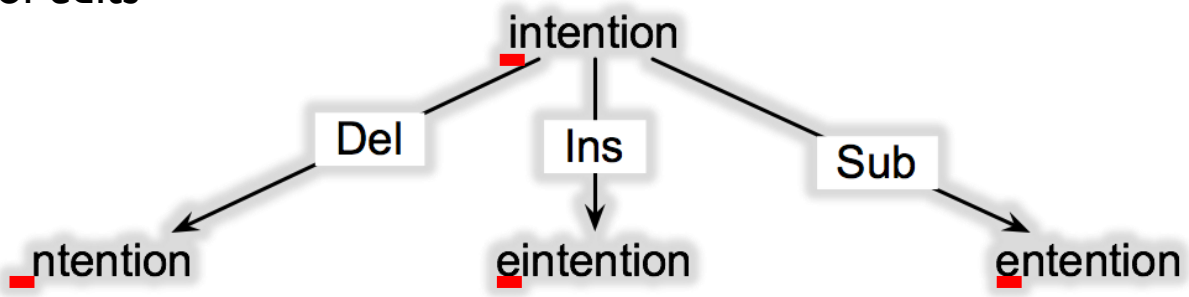
```
I N T E * N T I O N
| | | | | | | | | |
* E X E C U T I O N
d s s   i s
```

delete an i
substitute e for n
substitute x for t
insert c
substitute u for n

Also called an **alignment**:
a correspondence between substrings of the two sequences

How to find the Minimum Edit Distance?

- Searching for a shortest path (sequence of edits) from the start string to the final string:
  - Initial state: the word we're transforming, e.g., "intention"
  - Operators: insert, delete, substitute
  - Goal state: the word we're trying to get to, e.g., "execution"
  - Path cost: the number of edits

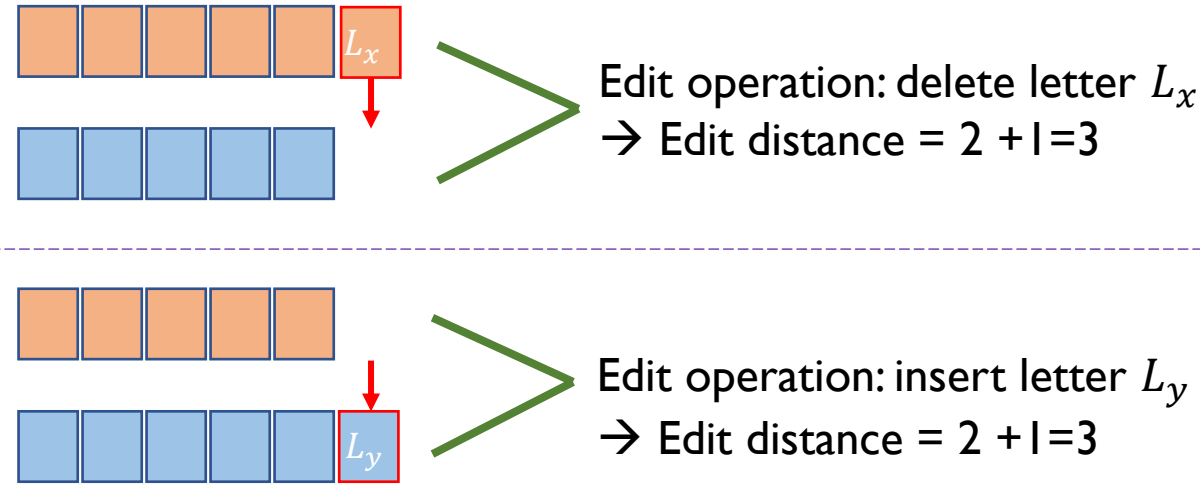


- The search space of all edit sequences is huge!
  - Lots of distinct paths wind up at the same state.
  - We don't have to keep track of all of them

A better way to search for shortest path(s)

- Assume we have computed the minimum edit distance between two words

- Then

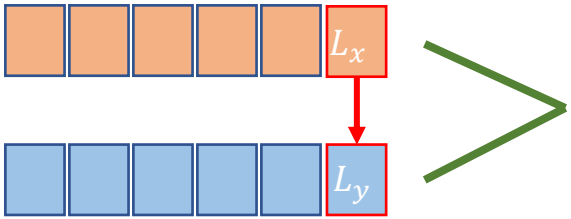


Edit distance increases by 1 for both cases regardless of which particular letter L_x or L_y is

A better way to search for shortest path(s)

- Assume we have computed the minimum edit distance between two words

- Then

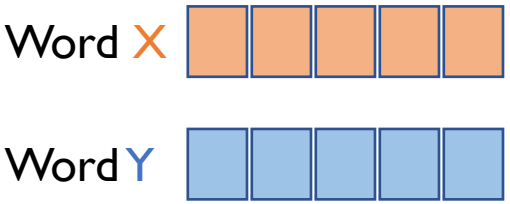


Two possible cases:

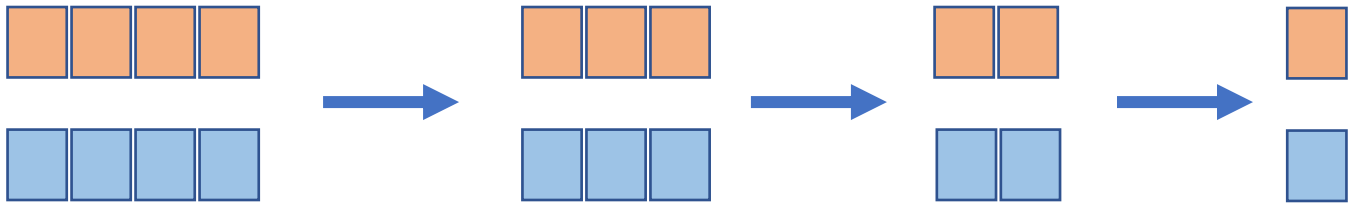
- If L_x and L_y are **different** letters,
  then edit distance = 2 +1=3 (substitute operation)
- If L_x and L_y are **the same** letter,
  then edit distance remains 2 (no edit operation needed)

A better way to search for shortest path(s)

- Then the problem becomes how to compute



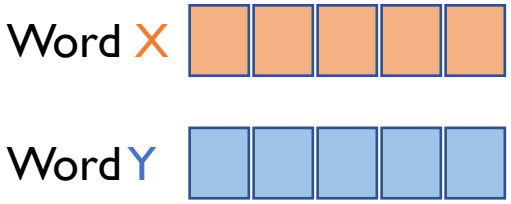
- Which can be computed if we know the edit distance of the 4-letter words, which can be computed if we know the ED of the 3-letter words…



- Summary: if we know how to compute 1-letter words, we can compute 2, 3, … n-letter words

Dynamic Programming for Minimum Edit Distance

- For two strings, X of length n letters, and Y of length m letters
  - We define $\boldsymbol{D(i,j)}$ to be 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 $\boldsymbol{D(n,m)}$



- Dynamic programming:
  - A tabular computation of $D(n,m)$
  - Solving problems by combining solutions to subproblems.
    - We compute $D(i,j)$ for small i,j , then compute larger $D(i,j)$ based on previously computed smaller values
    - We compute $D(i,j)$ for all i $(0 \leq i < n)$ and j $(0 \leq j < m)$
- We can assign different costs to different operations
  - ins-cost(.) =1, del-cost(.) = 1, sub-cost(.)=1
  - ins-cost(.) =1, del-cost(.) = 1, sub-cost(.)=2 (Levenshtein)

Computing Minimum Edit Distance

- Initialization
  - $D(i,0) = i$
  - $D(0,j) = j$

We define $\boldsymbol{D(i,j)}$ to be 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

- Recurrence Relation:

For each $i = 1 \ldots M$

For each $j = 1 \ldots N$

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

- Termination:
  - D(N,M) is distance

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

Dynamic Programming for Minimum Edit Distance

function MIN-EDIT-DISTANCE(*source*, *target*) **returns** *min-distance*

$n \leftarrow$ LENGTH(*source*)
$m \leftarrow$ LENGTH(*target*)
Create a distance matrix *D[n+1,m+1]*

 # Initialization: the zeroth row and column is the distance from the empty string
$D[0,0] = 0$
for each row i **from** 1 **to** n **do**
  $D[i,0] \leftarrow D[i\text{-}1,0] + del\text{-}cost(source[i])$
for each column j **from** 1 **to** m **do**
  $D[0,j] \leftarrow D[0,j\text{-}1] + ins\text{-}cost(target[j])$

 # Recurrence relation:
for each row i **from** 1 **to** n **do**
  for each column j **from** 1 **to** m **do**
    $D[i,j] \leftarrow$ MIN($D[i-1,j] + del\text{-}cost(source[i])$,
      $D[i-1,j-1] + sub\text{-}cost(source[i], target[j])$,
      $D[i,j-1] + ins\text{-}cost(target[j])$)
 # Termination
return D[n,m]

We define $\boldsymbol{D(i,j)}$ to be 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

Dynamic Programming for MED: Initialization

function MIN-EDIT-DISTANCE(*source*, *target*) **returns** *min-distance*

$n \leftarrow$ LENGTH(*source*)
$m \leftarrow$ LENGTH(*target*)
Create a distance matrix *D[n+1,m+1]*

 # Initialization: the zeroth row and column is the distance from the empty string
D[0,0] = 0
for each row *i* **from** 1 **to** *n* **do**
    D[i,0] $\leftarrow$ *D*[*i*-1,0] + *del-cost*(*source*[*i*])
for each column *j* **from** 1 **to** *m* **do**
    D[0,j] $\leftarrow$ *D*[0, *j*-1] + *ins-cost*(*target*[*j*])

intention

Target

Source

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

Computation

```
# Recurrence relation:
for each row i from 1 to n do
    for each column j from 1 to m do
        D[i,j] ← MIN( D[i−1,j] + del-cost(source[i]),
                      D[i−1,j−1] + sub-cost(source[i], target[j]),
                      D[i,j−1] + ins-cost(target[j]))
# Termination
return D[n,m]
```

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

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

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

Minimum Edit Distance and Alignment

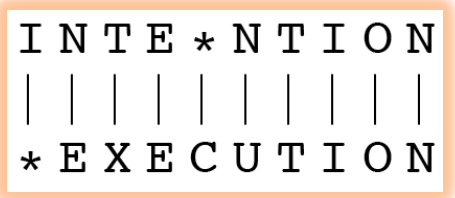
Src\Tar	#	e	x	e	c	u	t	i	o	n
#	0	1	2	3	4	5	6	7	8	9
i	1	2	3	4	5	6	7	6	7	8
n	2	3	4	5	6	7	8	7	8	7
t	3	4	5	6	7	8	7	8	9	8
e	4	3	4	5	6	7	8	9	10	9
n	5	4	5	6	7	8	9	10	11	10
t	6	5	6	7	8	9	8	9	10	11
i	7	6	7	8	9	10	9	8	9	10
o	8	7	8	9	10	11	10	9	8	9
n	9	8	9	10	11	12	11	10	9	8

We store **backpointers** in each cell. The backpointer (the arrows in the cells) from a cell *points to the previous cell* (or cells) that **we came from** in entering the current cell.

	#	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**

Computing alignments

- We often need to **align** each character of the two strings to each other
- We do this by keeping "backtrace" pointers
  - Every time we enter a cell, remember where we came from
  - Some cells have multiple backpointers because the minimum extension could have come from multiple previous cells.
- When we reach the end, trace back the path from the upper right corner to read off the alignment
  - In a backtrace, we start from the last cell, and follow the pointers back through the dynamic programming matrix.
  - Each complete path between the final cell and the initial cell is a minimum distance alignment.
  - There could be multiple paths, hence multiple alignments.



Summary

- Words and Corpora
  - Datasheet specifies properties of a dataset
  - Words: lemma, word forms
- Tokenization and normalization
  - Issues with tokenization
  - Case folding, lemmatization, stemming
  - Sentence segmentation
- Edit distance
  - Applications
  - Algorithm

- Reading: Chapter 2 https://web.stanford.edu/~jurafsky/slp3/

What can we do?

- Given a document we are able to segment its words and sentences.
  - The idea of word segmentation and sentence segmentation is similar, except the unit of processing is different, i.e., word vs sentence.
  - Depends on the characteristics of the document, we may need to select the most appropriate tokenizers.

- Given a word, we are able to perform normalization, to get the lemma or stem.

- Given two words, we are able to measure the similarity or distance between them, by Edit Distance
  - The same idea can be applied to measure two sentences, except the unit of processing is different, i.e., character vs word