Foundations of NLP

CS3126

Lecture-1

1.1 Regular Expressions

1.2 Text Normalization

1.3 Tokenization algorithms

Mahindra™ University

Global Thinkers. Engaged Leaders.

Acknowledgments

These slides were adapted from the book

SPEECH and LANGUAGE PROCESSING: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition and

Some modifications from presentations and resources found in the WEB by several scholars.

Recap

- Regular Expressions
- Applications
- Regex rules
- Class activity

Class Activity

Goal: To match regular expressions

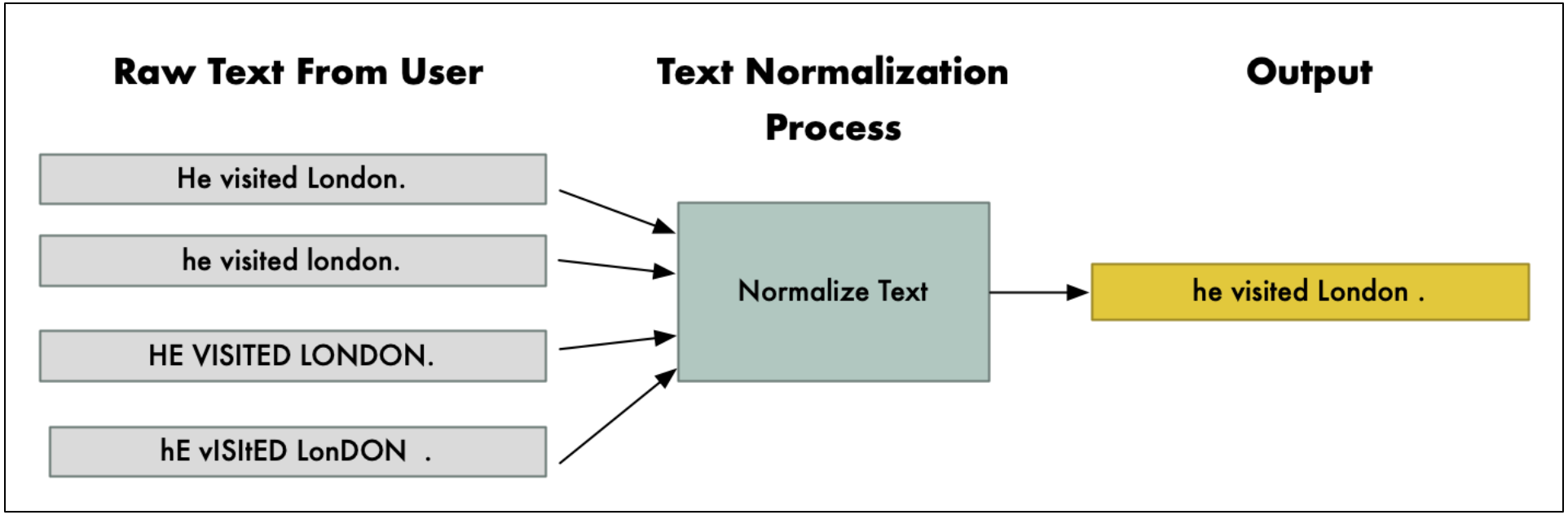
Task: To generate a large dataset of student records and write a Python program to match specific regular expressions against the data.

You will generate a dataset consisting of 100,000 student records. Each record will contain the following fields:

- **Student Name:** A random name composed of a first and last name.
- **Roll Number:** A unique identifier, such as SE22UARI001 (where "SE22UARI" is a batch code and the digits are a sequence).
- **Courses Taken:** A list of 3-5 courses represented by course codes like CS3126, CS3202, etc.
- **Email:** A randomly generated email address associated with the student with domain name @mahindrauniversity.edu.in.
- **Section:** The section AI1,AI2, AI3 etc.

Outcome: By the end of this activity, you should be able to: 1. Understand the use of regular expressions in data filtering. 2. Generate large datasets programmatically. 3. Apply regular expressions effectively to extract meaningful information from datasets.

Text Normalization



Slide Reference: 2_TextProc_Mar_25_2021.pdf (stanford.edu)

Real world issues in text that need Normalization

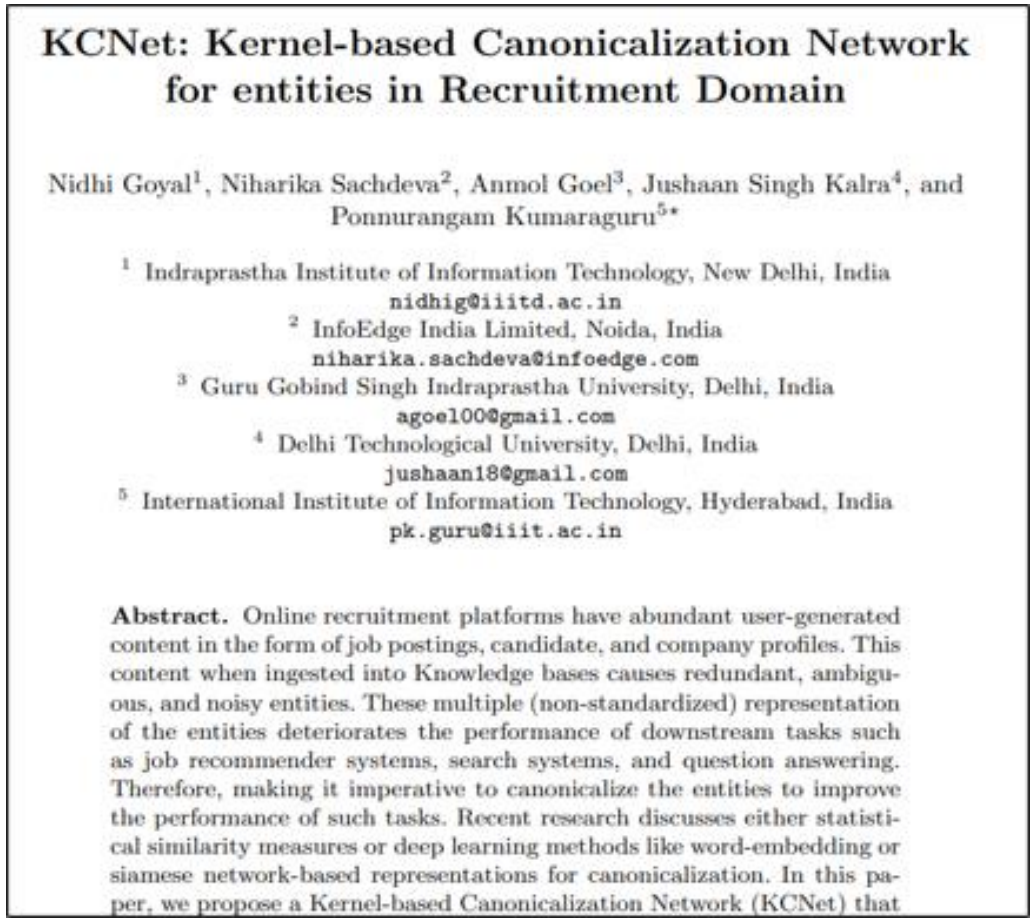
Industry examples:

Recruitment Domain

E-commerce

and many more.....

Research- Real-world/Industry use-case



KCNet: Kernel-based Canonicalization Network for entities in Recruitment Domain

Nidhi Goyal[1], Niharika Sachdeva[2], Anmol Goel[3], Jushaan Singh Kalra[4], and Ponnurangam Kumaraguru[5*]

[1] Indraprastha Institute of Information Technology, New Delhi, India
nidhig@iiitd.ac.in
[2] InfoEdge India Limited, Noida, India
niharika.sachdeva@infoedge.com
[3] Guru Gobind Singh Indraprastha University, Delhi, India
agoel00@gmail.com
[4] Delhi Technological University, Delhi, India
jushaan18@gmail.com
[5] International Institute of Information Technology, Hyderabad, India
pk.guru@iiit.ac.in

Abstract. Online recruitment platforms have abundant user-generated content in the form of job postings, candidate, and company profiles. This content when ingested into Knowledge bases causes redundant, ambiguous, and noisy entities. These multiple (non-standardized) representation of the entities deteriorates the performance of downstream tasks such as job recommender systems, search systems, and question answering. Therefore, making it imperative to canonicalize the entities to improve the performance of such tasks. Recent research discusses either statistical similarity measures or deep learning methods like word-embedding or siamese network-based representations for canonicalization. In this paper, we propose a Kernel-based Canonicalization Network (KCNet) that

https://cdn.iiit.ac.in/cdn/precog.iiit.ac.in/pubs/2021_July_KCNet-slides.pdf

Basic Normalization steps:

1. Segmenting/tokenizing words in running text
2. Normalizing word formats
3. Segmenting sentences in running text

Slide Reference: 2_TextProc_Mar_25_2021.pdf (stanford.edu)

How many words in a sentence?

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?

How many words in a sentence?

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) (or 11?)

How many words in a corpus?

N = number of tokens

V = vocabulary = set of types

|V| is size of vocabulary

Heaps law/Herden's law

N = number of tokens

V = vocabulary = set of types

|V| is size of vocabulary

Heaps Law = Herdan's Law = |V|= $= kN^b$ where often .67 < β < .75

i.e., vocabulary size grows with > square root of the number of word tokens

Heaps' law: Estimating the number of terms (stanford.edu)

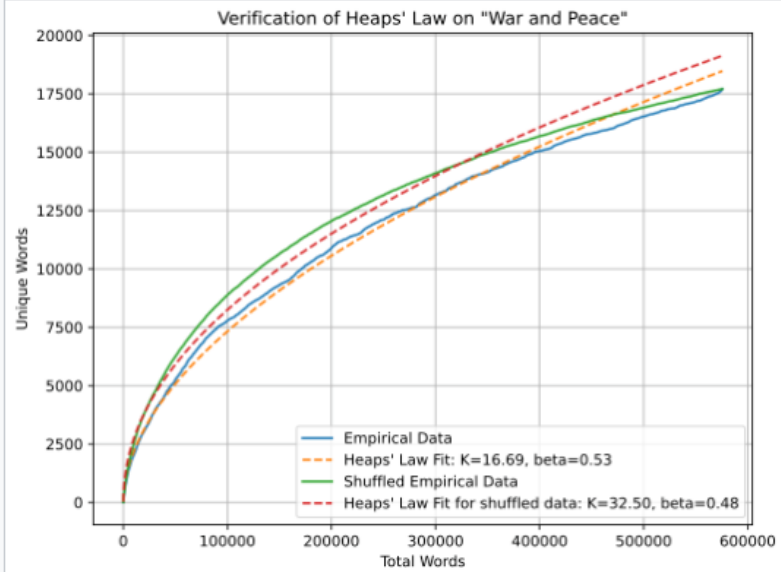
Heaps law/Herden's law

From Wikipedia, the free encyclopedia

In linguistics, **Heaps' law** (also called **Herdan's law**) is an empirical law which describes the number of distinct words in a document (or set of documents) as a function of the document length (so called type-token relation). It can be formulated as

$$V_R(n) = Kn^{\beta}$$

where V_R is the number of distinct words in an instance text of size n. K and β are free parameters determined empirically. With English text corpora, typically K is between 10 and 100, and β is between 0.4 and 0.6.



Verification of Heaps' law on *War and Peace*, as well as a randomly shuffled version of it. Both cases fit well to the Heaps' law with very similar exponents β, but different K.

Heaps' law - Wikipedia

How many words in a corpus?

	Tokens = N	Types = |V|
Switchboard phone conversations	2.4 million	20 thousand
Shakespeare	884,000	31 thousand
COCA	440 million	2 million
Google N-grams	1 trillion	13+ million

Corpora

Words don't appear out of nowhere! A text is produced by

- a specific writer(s),
- at a specific time,
- in a specific variety,
- of a specific language,
- for a specific function.

Corpora vary along dimension like

- **Language**: 7097 languages in the world
- **Variety**, like African American Language varieties.
  - AAE Twitter posts might include forms like "*iont" (I don't)*
- **Code switching**, e.g., Spanish/English, Hindi/English:

  S/E: Por primera vez veo a @username actually being hateful! It was beautiful:)

  [For the first time I get to see @username actually being hateful! it was beautiful:)]

  H/E: dost tha or ra- hega ... dont wory ... but dherya rakhe

  [“he was and will remain a friend ... don’t worry ... but have faith”]
- **Genre:** newswire, fiction, scientific articles, Wikipedia
- **Author Demographics**: writer's age, gender, ethnicity, SES

Tokenization

Input: Mahindra university department

Tokens:

Mahindra

University

Department

A token is a sequence of characters in a document

Space-based tokenization

A very simple way to tokenize

- For languages that use space characters between words Arabic, Cyrillic, Greek, Latin, etc., based writing systems

- Segment off a token between instances of spaces

Simple Tokenization in UNIX

(Inspired by Ken Church's UNIX for Poets.)

Given a text file, output the word tokens and their frequencies

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

Change all non-alpha to newlines

Sort in alphabetical order

Merge and count each type

```
1945 A
  72 AARON
  19 ABBESS
   5 ABBOT
 ... ...
```

```
25 Aaron
 6 Abate
 1 Abates
 5 Abbess
 6 Abbey
 3 Abbot
.... …
```

The first step: tokenizing

```
tr -sc 'A-Za-z' '\n' < shakes.txt | head
```

```
THE
SONNETS
by
William
Shakespeare
From
fairest
creatures
We
...
```

The second step: sorting

```
tr -sc 'A-Za-z' '\n' < shakes.txt | sort | head
```

```
A
A
A
A
A
A
A
A
A
...
```

More counting

Merging upper and lower case

```
tr 'A-Z' 'a-z' < shakes.txt | tr -sc 'A-Za-z' '\n' | sort | uniq -c
```

Sorting the counts

```
tr 'A-Z' 'a-z' < shakes.txt | tr -sc 'A-Za-z' '\n' | sort | uniq -c | sort -n -r
```

```
23243 the
22225 i
18618 and
16339 to
15687 of
12780 a
12163 you
10839 my
10005 in
```

What happened here?

Issues in Tokenization

Can't just blindly remove punctuation:

- m.p.h., Ph.D., AT&T, cap’n
- prices ($45.55)
- dates (01/02/06)
- URLs (http://www.stanford.edu)
- hashtags (#nlproc)
- email addresses (someone@cs.colorado.edu)

Clitic: a word that doesn't stand on its own

- "are" in we're, French "je" in j'ai, "le" in l'honneur

When should multiword expressions (MWE) be words?

- New York, rock ’n’ roll

What are valid tokens?

Hewlett-Packard Company

Are these two tokens "Hewlett" or "Packard" or one token?

Mahindra university -> 1 token or two?

State-of-the-art-> how many tokens?

Language issues-> Left-right or right-left (For example: Arabic)

Simple Code Example (Python 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, 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', '...']
```

Source: https://web.stanford.edu/~jurafsky/slp3/2.pdf

Tokenization in languages without spaces

Many languages (like Chinese, Japanese, Thai) don't use spaces to separate words!

How do we decide where the token boundaries should be?

Word tokenization in Chinese

- Chinese words are composed of characters called "**hanzi**" (or sometimes just "**zi**")
- Each one represents a meaning unit called a morpheme. Each word has on average 2.4 of them.
- But deciding what counts as a word is complex and not agreed upon.

How to do word tokenization in Chinese?

姚明进入总决赛 “Yao Ming reaches the finals”

How to do word tokenization in Chinese?

姚明进入总决赛 “Yao Ming reaches the finals”

3 words?
姚明 进入 总决赛
YaoMing reaches finals

How to do word tokenization in Chinese?

姚明进入总决赛 “Yao Ming reaches the finals”

3 words?

姚明	进入	总决赛
YaoMing	reaches	finals

5 words?

姚	明	进入	总	决赛
Yao	Ming	reaches	overall	finals

How to do word tokenization in Chinese?

姚明进入总决赛 “Yao Ming reaches the finals”

3 words?
姚明 进入 总决赛
YaoMing reaches finals

5 words?
姚 明 进入 总 决赛
Yao Ming reaches overall finals

7 characters? (don't use words at all):
姚 明 进 入 总 决 赛
Yao Ming enter enter overall decision game

Word tokenization / segmentation

- So in Chinese it's common to just treat each character (zi) as a token.
  - So the **segmentation** step is very simple
- In other languages (like Thai and Japanese), more complex word segmentation is required.
  - The standard algorithms are neural sequence models trained by supervised machine learning.

Another option for text tokenization

Instead of

- white-space segmentation
- single-character segmentation

Use the data to tell us how to tokenize.

Subword tokenization (because tokens can be parts of words as well as whole words)

Complexity in Word tokenization

Word tokenization is more complex in languages like written Chinese, Japanese, and Thai, which do not use spaces to mark potential word-boundaries

Another Solution-

Byte-pair encoding – [Read the example from book]

Subword tokenization

Three common algorithms:

- **Byte-Pair Encoding (BPE)** (Sennrich et al., 2016)
- **Unigram language modeling tokenization** (Kudo, 2018)
- **WordPiece** (Schuster and Nakajima, 2012)

All have 2 parts:

- A token **learner** that takes a raw training corpus and induces a vocabulary (a set of tokens).
- A token **segmenter** that takes a raw test sentence and tokenizes it according to that vocabulary

Byte Pair Encoding (BPE) token learner

Let vocabulary be the set of all individual characters

= {A, B, C, D,…, a, b, c, d….}

Repeat:

- Choose the two symbols that are most frequently adjacent in the training corpus (say 'A', 'B')
- Add a new merged symbol 'AB' to the vocabulary
- Replace every adjacent 'A' 'B' in the corpus with 'AB'.

Until *k* merges have been done.

BPE token learner algorithm

function BYTE-PAIR ENCODING(strings C, number of merges k) **returns** vocab V

$V \leftarrow$ all unique characters in C # initial set of tokens is characters
for $i = 1$ **to** k **do** # merge tokens til k times
  $t_L, t_R \leftarrow$ Most frequent pair of adjacent tokens in C
  $t_{NEW} \leftarrow t_L + t_R$ # make new token by concatenating
  $V \leftarrow V + t_{NEW}$ # update the vocabulary
  Replace each occurrence of t_L, t_R in C with t_{NEW} # and update the corpus
return V

Byte Pair Encoding (BPE) Addendum

Most subword algorithms are run inside space-separated tokens.

So we commonly first add a special end-of-word symbol '__' before space in training corpus

Next, separate into letters.

BPE token learner

Original (very fascinating 🙄) corpus:

low low low low low lowest lowest newer newer newer newer newer newer wider wider wider new new

Add end-of-word tokens, resulting in this vocabulary:

vocabulary

_, d, e, i, l, n, o, r, s, t, w

BPE token learner

corpus	vocabulary
5 l o w _	_, d, e, i, l, n, o, r, s, t, w
2 l o w e s t _	
6 n e w e r _	
3 w i d e r _	
2 n e w _	

Merge e r to er

corpus	vocabulary
5 l o w _	_, d, e, i, l, n, o, r, s, t, w, er
2 l o w e s t _	
6 n e w er _	
3 w i d er _	
2 n e w _	

BPE

corpus

5 l o w _
2 l o w e s t ___
6 n e w er __
3 w i d er __
2 n e w __

vocabulary

__, d, e, i, l, n, o, r, s, t, w, er

Merge er _ to er_

corpus

5 l o w ____
2 l o w e s t _____
6 n e w er___
3 w i d er____
2 n e w ___

vocabulary

__, d, e, i, l, n, o, r, s, t, w, er, er____

BPE

corpus	vocabulary
5 l o w _	_, d, e, i, l, n, o, r, s, t, w, er, er_____
2 l o w e s t ___	
6 n e w er___	
3 w i d er___	
2 n e w _	

Merge n e to ne

corpus	vocabulary
5 l o w _____	_, d, e, i, l, n, o, r, s, t, w, er, er_____, ne
2 l o w e s t ___	
6 ne w er_____	
3 w i d er_____	
2 ne w ____	

BPE

The next merges are:

Merge	Current Vocabulary
(ne, w)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new
(l, o)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo
(lo, w)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low
(new, er _)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer _
(low, _)	_, d, e, i, l, n, o, r, s, t, w, er, er_, ne, new, lo, low, newer _, low_

BPE token segmenter algorithm

On the test data, run each merge learned from the training data:

- Greedily
- In the order we learned them
- (test frequencies don't play a role)

So: merge every e r to er, then merge er _ to er_, etc.

Result:

- Test set "n e w e r _" would be tokenized as a full word
- Test set "l o w e r _" would be two tokens: "low er_"

Properties of BPE tokens

Usually include frequent words

And frequent subwords

- Which are often morphemes like *-est* or *–er*

A **morpheme** is the smallest meaning-bearing unit of a language

- *unlikeliest* has 3 morphemes *un-*, *likely*, and *-est*

Sentence Segmentation

!, ? mostly unambiguous but **period** "." is very ambiguous

- Sentence boundary
- Abbreviations like Inc. or Dr.
- Numbers like .02% or 4.3

Common algorithm: Tokenize first: use rules or ML to classify a period as either

(a) part of the word or (b) a sentence-boundary.

- An abbreviation dictionary can help

Sentence segmentation can then often be done by rules based on this tokenization.

Implementation/Tokenization

https://huggingface.co/learn/nlp-course/en/chapter6/5

https://github.com/SumanthRH/tokenization

Class Activity

- Implement Byte Pair Encoding Algorithm from scratch and use the below corpus:

fast-bpe/tinyshakespeare.txt at main · IAmPara0x/fast-bpe (github.com)

Other tokenizers

- Word piece tokenizers [https://huggingface.co/learn/nlp-course/chapter6/6]
- Sentence piece tokenizers [https://github.com/google/sentencepiece]

Word Normalization

Putting words/tokens in a standard format

- U.S.A. or USA
- uhhuh or uh-huh
- Fed or fed
- am, is, be, are

Case folding

- Applications such as Information retrieval: reduce all letters to lower case
  - Since all users tend to use lower case
  - Possible exceptions: uppercase in mid-sentence?

Case folding

- Applications such as Information retrieval: reduce all letters to lower case
  - Since all users tend to use lower case
  - Possible exceptions: uppercase in mid-sentence?

  - General Motors
  - Fed vs. Fed
  - SAIL vs. sail

Case folding

- Applications such as Information retrieval: reduce all letters to lower case
  - Since all users tend to use lower case
  - Possible exceptions: uppercase in mid-sentence?

  - General Motors
  - Fed vs. Fed
  - SAIL vs. Sail
- Sentiment analysis (US and us have different meanings)

Morphology

- Morphemes:
  - The small meaningful units that make up words
  - **Stems**: The core meaning-bearing units
  - **Affixes**: Parts that adhere to stems, often with grammatical functions
- Morphological Parsers:
  - Parse *cats* into two morphemes *cat* and *s*
  - Parse Spanish *amaren* ('if in the future they would love') into morpheme *amar* 'to love', and the morphological features *3PL* and *future subjunctive*.

Dealing with complex morphology is necessary for many languages

- e.g., the Turkish word:
- 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'

Objective

- The goal of both stemming and lemmatization is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form.

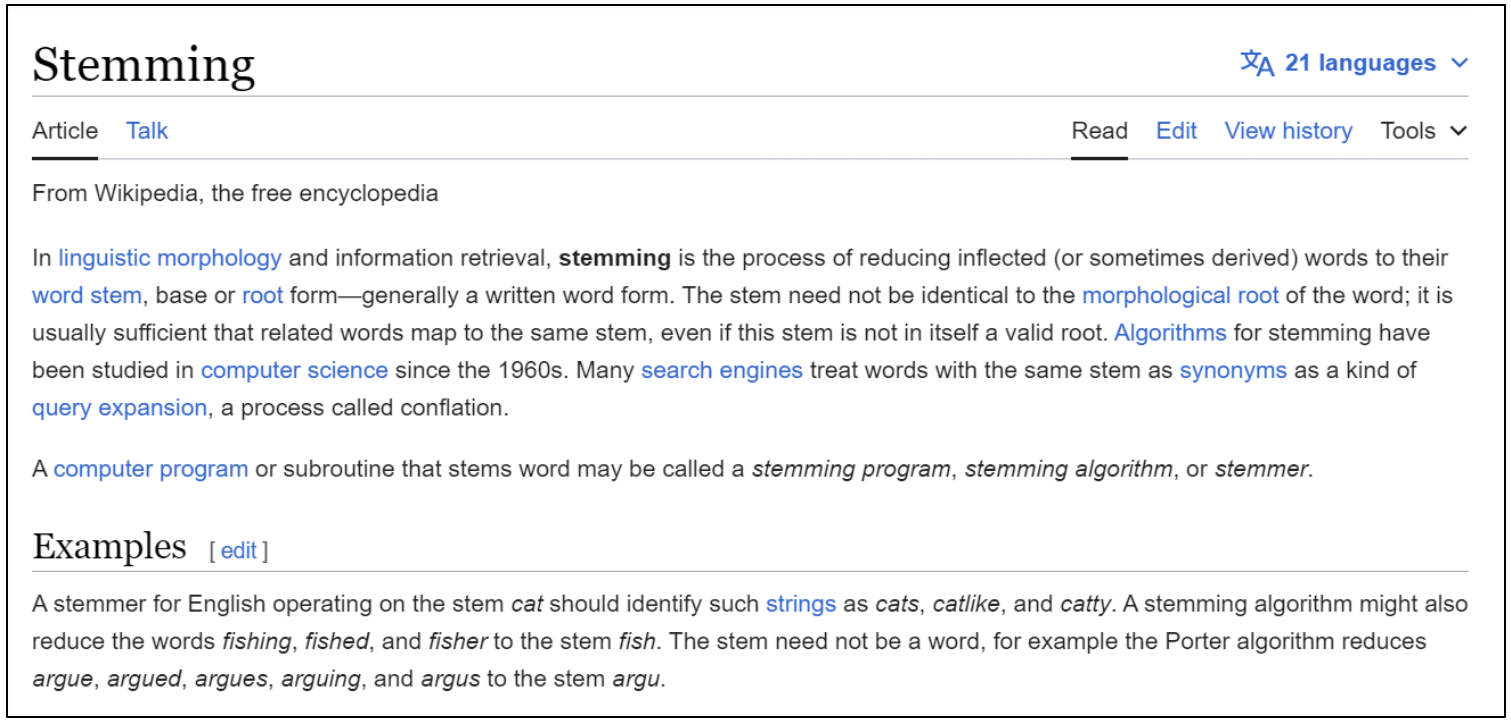
For instance:

- am, are, is be
  car, cars, car's, cars' car
- The result of this mapping of text will be something like:
  - the boy's cars are different colors
  - the boy car be differ color

Stemming in NLP

affect	amus	close
affect	amuse	close
affectation	amused	closed
affected	amusement	closely
affecting	amusements	closing
affection	amusing	grate
affections		grate
affects		grateful
		gratefully

Stemming



Stemming

21 languages

Article Talk Read Edit View history Tools

From Wikipedia, the free encyclopedia

In linguistic morphology and information retrieval, **stemming** is the process of reducing inflected (or sometimes derived) words to their word stem, base or root form—generally a written word form. The stem need not be identical to the morphological root of the word; it is usually sufficient that related words map to the same stem, even if this stem is not in itself a valid root. Algorithms for stemming have been studied in computer science since the 1960s. Many search engines treat words with the same stem as synonyms as a kind of query expansion, a process called conflation.

A computer program or subroutine that stems word may be called a *stemming program*, *stemming algorithm*, or *stemmer*.

Examples [edit]

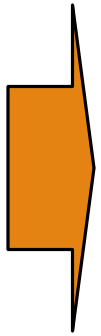
A stemmer for English operating on the stem *cat* should identify such strings as *cats*, *catlike*, and *catty*. A stemming algorithm might also reduce the words *fishing*, *fished*, and *fisher* to the stem *fish*. The stem need not be a word, for example the Porter algorithm reduces *argue*, *argued*, *argues*, *arguing*, and *argus* to the stem *argu*.

https://en.wikipedia.org/wiki/Stemming#

Stemming

Reduce terms to stems, chopping off affixes crudely

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.



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 .

Stemming

Stemming suggests crude affix chopping

- language dependent
- automation, automatic, automate ---(automat)

Stemming programs are called as Stemmers or Stemming algorithms

Porter Stemming Algorithm (tartarus.org)

The Porter Stemmer (Porter, 1980)

- Common Algorithm for English language
- A simple rule-based algorithm for stemming
- An example of a HEURISTIC method
- Based on rules like:
  - ATIONAL -> ATE (e.g., rel**ational** -> rel**ate**)
- The algorithm consists of 7 sets of rules, applied in order

tartarus.org/martin/PorterStemmer/def.txt

The Porter Stemmer: definitions

- Definitions:
  - **CONSONANTS**: a letter other than A, E, I, O, U, and Y preceded by consonant
  - **VOWEL**: any other letter (if the letter is not a consonant)
- With this definition, all words are of the form: $(C)(VC)^m(V)$
  - C: string of one or more consonants (con+)
  - V: string of one or more vowels
  - m: measure of word or word part which is represented in form of VC
- E.g.
  - **Troubles**
  - C $(VC)^m$ V

tartarus.org/martin/PorterStemmer/def.txt

Measure of the word

- M=0 TREE, BY, TR
- M=1 TROUBLE, OATS, TREES, IVY
- M=2 TROUBLES, PRIVATE, OATEN

tartarus.org/martin/PorterStemmer/def.txt

The Porter Stemmer: Rule format

- The rules are of the form:
- (condition) S1 -> S2 where S1 and S2 are suffixes
- If the rule (m>1) EMENT->
  - In this S1 is EMENT and S2 is NULL
  - So, this would map REPLACEMENT with REPLAC

tartarus.org/martin/PorterStemmer/def.txt

Conditions

m	The measure of the stem
***S**	The stem ends with S
v	The stem contains a vowel
***d**	The stem ends with a double consonant (TT,SS)
***o**	The stem ends in CV C (second C not W, X, or Y) Ex: WIL, HOP

The condition may also contains expressions with and, or, or not

Example ((m>1) and (*s or*t)) -tests for a stem with m>1 ending in s or t

tartarus.org/martin/PorterStemmer/def.txt

The Porter Stemmer: Step 1

- SSES -> SS
  - caresses -> caress
- IES -> I
  - ponies -> poni
  - ties -> ti
- SS -> SS
  - caress -> caress
- S -> €
  - cats -> cat

tartarus.org/martin/PorterStemmer/def.txt

The Porter Stemmer: Step 2a (past tense, progressive)

- (m>0) EED -> EE
  - **Condition verified**: agreed -> agree
  - **Condition not verified**: feed -> feed
- (*V*) ED -> €
  - **Condition verified**: plastered -> plaster
  - **Condition not verified**: bled -> bled
- (*V*) ING -> €
  - **Condition verified:** motoring -> motor
  - **Condition not verified**: sing -> sing

tartarus.org/martin/PorterStemmer/def.txt

The Porter Stemmer: Step 2b (cleanup)

- (These rules are ran if second or third rule in 2a apply)
- AT -> ATE
  - Conflat(ed) -> conflate
- BL -> BLE
  - Troubl(ing) - > trouble
- (*d & ! (*L or *S or *Z)) -> single letter
  - **Condition verified**: hopp(ing) -> hop, tann(ed) -> tan
  - **Condition not verified**: fall(ing) -> fall
- (m=1 & *o) -> E
  - **Condition verified**: fil(ing) -> file
  - Condition not verified: fail -> fail

tartarus.org/martin/PorterStemmer/def.txt

The Porter Stemmer: step 3 and 4

- Step 3: Y elimination (*V*) Y -> I
  - **Condition verified**: happy -> happi
  - **Condition not verified**: sky -> sky
- Step 4: Derivational Morphology, I
  - (m>0) ATIONAL -> ATE
    - Relational -> relate
  - (m>0) IZATION -> IZE
    - Generalization -> generalize
  - (m>0) BILITI -> BLE
    - Sensibiliti -> sensible

tartarus.org/martin/PorterStemmer/def.txt

Porter Stemmer Step 5 and Step 6

- Derivational Morphology II
  - (m>0) ICATE-> IC
    - Triplicate-> Triplic
  - (m>0) FUL -> €
    - hopeful-> hope
  - (m>0) NESS-> €
    - goodness->good
- Derivational Morphology III
  - (m>1) ANCE-> €
    - allowance-> allow
  - (m>1) ENT -> €
    - dependent-> depend
  - (m>1) IVE-> €
    - effective->effect

tartarus.org/martin/PorterStemmer/def.txt

The porter stemmer Step 7 (cleanup)

- Step 7a
  - (m>1) E -> €
    - Probate -> probat
  - (m=1 & !*o) NESS -> €
    - Goodness -> good
- Step 7 b
  - (m>1 & *d & *L) -> single letter
    - **Condition verified**: controll -> control
    - **Condition not verified:** roll -> roll

tartarus.org/martin/PorterStemmer/def.txt

Advantages

- **Speed and efficiency**: Stemming algorithms are generally faster as they follow simple rule-based approaches.
- **Simplicity:** The algorithms for stemming use simple heuristic rules, so they are less complex to implement and understand than other methods.
- **Improved search performance**: In search engines and information retrieval systems, stemming helps connect different word forms, potentially increasing the breadth of search results.

Disadvantages

- **Over-stemming and under-stemming**:

Stemming can often be imprecise, leading to over-stemming (where words are overly reduced and unrelated words are conflated) and under-stemming (where related words don’t appear related).

- **Language limitations:**

The effectiveness of a stemming algorithm reduces if words appear in irregular formats (i.e., irregular conjugated forms).

Lemmatization

- Lemmatization goes beyond truncating words and analyzes the context of the sentence, considering the word's use in the larger text and its inflected form.

- After determining the word's context, the lemmatization algorithm returns the word's base form (lemma) from a dictionary reference.

- Task of determining whether two words have same root despite surface differences

Lemmatization

Lemmatization

18 languages

Article Talk Read Edit View history Tools

From Wikipedia, the free encyclopedia

Lemmatization (or less commonly **lemmatisation**) in linguistics is the process of grouping together the inflected forms of a word so they can be analysed as a single item, identified by the word's lemma, or dictionary form.[1]

In computational linguistics, lemmatization is the algorithmic process of determining the lemma of a word based on its intended meaning. Unlike stemming, lemmatization depends on correctly identifying the intended part of speech and meaning of a word in a sentence, as well as within the larger context surrounding that sentence, such as neighbouring sentences or even an entire document. As a result, developing efficient lemmatization algorithms is an open area of research.[2][3][4]

Description [edit]

In many languages, words appear in several *inflected* forms. For example, in English, the verb 'to walk' may appear as 'walk', 'walked', 'walks' or 'walking'. The base form, 'walk', that one might look up in a dictionary, is called the *lemma* for the word. The association of the base form with a part of speech is often called a *lexeme* of the word.

https://en.wikipedia.org/wiki/Lemmatization

Lemmatization

- The most sophisticated methods for lemmatization involve complete **morphological parsing** of the word.
- Morphology is the study of morpheme the way words are built up from smaller meaning-bearing units called **morphemes**.
- Two broad classes of morphemes can be distinguished:
  - **stems**—the central morpheme of the word, supplying the main meaning— and **affixes**—adding "additional" meanings of various kinds.
  - So, for example, the word **fox** consists of one morpheme (the morpheme **fox**) and the word **cats** consists of two: the morpheme **cat** and the morpheme **-s**.

Lemmatization

Represent all words as their lemma, their shared root= dictionary headword form:

- *am, are, is* → *be*
- *car, cars, car's, cars'* → *car*
- Spanish quiero (‘I want’), quieres (‘you want’) → querer ‘want'
- *He is reading detective stories* → *He be read detective story*

Lemmatization: Advantages

Accuracy and contextual understanding: Lemmatization is more accurate as it considers words' context and the morphological analysis. It can distinguish between different word uses based on its part of speech.

Reduced ambiguity: By converting words to their dictionary form, lemmatization reduces ambiguity and enhances the clarity of text analysis.

Language and grammar compliance: Lemmatization adheres more closely to the grammar and vocabulary of the target language, leading to linguistically meaningful outputs.

Lemmatization: Disadvantages

Computational complexity:

Lemmatization algorithms are more complex and computationally intensive than stemming. They require more processing power and time.

Dependency on language resources:

Lemmatization depends on extensive language-specific resources like dictionaries and morphological analyzers, making it less flexible for use with certain languages, such as Arabic.

Stemming vs Lemmatization

Stemming

achieve -> achiev
achieving -> achiev

- Can reduce words to a stem that is not an existing word
- Operates on a single word without knowledge of the context
- Simpler and faster

Lemmatization

achieve -> achieve
achieving -> achieve

- Reduces inflected words to their lemma, which is always an existing word
- Can leverage context to find the correct lemma of a word
- More accurate but slower

In-class activity

Exercise1:

- Convert these list of words into base form using Stemming and Lemmatization and observe the transformations
  - ['running', 'painting', 'walking', 'dressing', 'likely', 'children', 'whom', 'good', 'ate', 'fishing']

- Write a short note on the words that have different base words using stemming and Lemmatization

In class activity

Write a python code to use NLTK library and convert the base forms using different Stemmers and Lemmatizers

- use different stemmers and lemmatizers provided by NLTK
- see https://www.nltk.org/howto/stem.html for full NLTK stemmer module documentations

Reference materials

- https://vlanc-lab.github.io/mu-nlp-course/teachings/fall-2024-AI-nlp.html

- Lecture notes

- (A) Speech and Language Processing by Daniel Jurafsky and James H. Martin
- (B) Natural Language Processing with Python. (updated edition based on Python 3 and NLTK
- 3) Steven Bird et al. O'Reilly Media