Basic Text Processing

Word tokenization

Text Preprocessing

Every NLP task needs to do text preprocessing (normalization, tokenization)

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

How many words?

Word: AABBZX

Distinct charc: 4, num of charc:6

|v|=4, N=6

Vocab: it will have unique words, also referred to as type

Token: all the occurrences of words

These types are features of interest for our NLP task

How many words?

they lay back on the San Francisco grass and looked at the stars and their

Type: an element of the vocabulary: num of distinct words/type define our vocabulary

Token: num of times those words occur are tokens

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

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

Morphology

Study of word forms, their construction and uses

Cat, cats

Run, ran, running; stop, stopped

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

Morphology

Morpheme: smallest unit that has some function in a word

Stem/root: core meaning bearing morpheme

Free morpheme can be used by itself

Cat; stop; run; sit

Affixes: bound morpheme: has to be bound with others to make sense

prefixes (un, dis, in, im) and suffixes (s, er, ly)

<u>Un</u>kind, <u>im</u>mature, <u>dis</u>like

Cat<u>s</u>; low<u>er</u>, like<u>ly</u>

Derivational: when they use with a free morpheme they <u>change its meaning</u>, or its part of speech (grammatical category)

. unkind, immature, teach" → "teacher" (suffix "-er" changes verb to noun)

Inflectional: <u>depicts plurality</u>, tense change, They do not change the word's core meaning or its part of speech

cats, stopped , running, ran is also an inflected form

1.Compounding:

1. **Process:** Compounding involves combining two or more independent words to create a new word.
2. **Formation:** The resulting compound word often carries a meaning that is a combination of the meanings of its individual components.
3. **Examples:** "toothbrush" (tooth + brush), "whiteboard" (white + board), "football" (foot + ball).

2.Reduplication:

1. **Process:** Reduplication involves the repetition of all or part of a word to create a new word or form.
2. **Formation:** The repeated portion can be a full or partial copy of the original word, and it often serves a grammatical or semantic function.
3. **Examples:**
    1. Full reduplication: "bye-bye," "choo-choo."
    2. Partial reduplication: "zigzag," "mishmash."

He sat on the chair but he likes sitting on the floor

Words are defined as space delimited sequence of characters

Tokens: N=12,

V{He, sat, on, the, chair, but, he, likes, sitting, floor}

He, he: should these be collapsed as one?

Normalization: He,he are same types (collapse them into 1)

Depends on the application you are working on

Sentiment analysis:

The food was delicious.

The food was DELICIOUSSS.

How many words? (Normalization)

sat, sitting : should these be collapsed as 1

Stemming: applies a set of rules to reduce words to their stems

Runner, run, ran, running, runs → run

The rules are applied recursively to reach the stem

Using stemmer you reduce the vocab

Automatic, automated, automata might be collapsed as **automat**

FP and FN errors are common in stemmers

Issues: might get meaningless words (automat),

might intermingle multiple meaning words into one (automatic, automata)

Stemming

Stemming algorithms can be **rule-based or probabilistic**.

Rule-based stemmers use a set of predefined rules to remove suffixes and other word endings. For example, a rule-based stemmer might remove the suffix -s from all nouns and the suffix -ed from all verbs.

- faster and simpler to implement, but they can be less accurate

Probabilistic stemmers use statistical models to determine the most likely stem for a word based on its frequency and usage patterns in a large corpus of text.

- use the frequency of word forms in a corpus to decide whether a word ending in -ing should be reduced to its base form (e.g., "running" to "run").
- These tend to be more accurate because they leverage statistical data to make more informed decisions. However, they require a large corpus for training and can be more computationally intensive.

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

A **wordform** is the specific, inflected form of a word as it appears in text.

It includes all variations of a word, such as different tenses, numbers (singular/plural), cases, etc.

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)

How many words? (Normalization)

Lemmatization: dictionary based

Replaces words with the dictionary root word/head word

*Lemmatization is the process of reducing a word to its base or dictionary form, called the **lemma**. This is different from stemming, which reduces words to their root form, which may not necessarily be a dictionary word.*

ran, runs, running → **run**

runner, runners → runner

sat, sitting → sit

Children → child

Normalization will be done according to your task!

Lemmatization

Reduce inflections or variant forms to base form

am, are, is $\rightarrow$ *be*

car, cars, car's, cars' $\rightarrow$ *car*

the boy's cars ***are*** *different colors* $\rightarrow$ *the boy car* ***be*** *different color*

Lemmatization: have to find correct dictionary headword form

Machine translation

Spanish quiero ('I want'), quieres ('you want') same lemma as querer 'want'

- **Lemmatization:**
  - Choose lemmatization when word accuracy and valid words are crucial.
  - Useful in applications where linguistic precision is required, such as sentiment analysis, machine translation, or question answering.
- **Stemming:**
  - Choose stemming when you need a faster, simpler approach.
  - Suitable for tasks like information retrieval, search engines, or cases where linguistic precision is less critical.

Porter's algorithm

The most common English stemmer

Step 1a

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

Step 1b

(*v*)ing → ø	walking	→ walk
	sing	→ sing
(*v*)ed → ø	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 → ø	revival	→ reviv
able → ø	adjustable	→ adjust
ate → ø	activate	→ activ

…

Porter stemming is more commonly applied to words with suffixes, such as plurals or verb inflections.

Verb inflections refer to changes in the form of a verb to convey different grammatical meanings. These changes typically involve variations in tense, mood, aspect, number, and person.

Viewing morphology in a corpus

(*v*)ing → ø walking → walk

sing → sing

Viewing morphology in a corpus Why only strip –ing if there is a vowel?

```
(*v*)ing → ø   walking   → walk
               sing      → sing
```

```
tr -sc 'A-Za-z' '\n' < shakes.txt | grep ’ing$' | sort | uniq -c | sort –nr
```

```
1312 King
 548 being
 541 nothing
 388 king
 375 bring
 358 thing
 307 ring
 152 something
 145 coming
 130 morning
```

```
548 being
541 nothing
152 something
145 coming
130 morning
122 having
120 living
117 loving
116 Being
102 going
```

```
tr -sc 'A-Za-z' '\n' < shakes.txt | grep '[aeiou].*ing$' | sort | uniq -c | sort –nr
```

Dealing with complex morphology is sometimes necessary

Some languages requires complex morpheme segmentation

- 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 'p1pl' + dan 'abl'
  - + mis 'past' + siniz '2pl' + casina 'as if'

Basic Text Processing

Word Normalization and Stemming

Stop Words

Do not play much role

Depending on your task, prepare a list of such words

On, the, a, an etc.

Why removing stop words is necessary?

Example

He sat on the chair but he likes sitting on the floor

Remove stop words (on, the) but retain sequence, why?

Collapse words to their stems

he sit <sw> <sw> chair but he like sit <sw> <sw> floor

|v|= 7 (how?), N= 12

If your analysis is based on vectors where sequence is not important, then you don’t need to retain the sequence of words.

<UNK> for unknown word not in the vocab

<DATE> for dates

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
8954  d
```

What happened here?

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**

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

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

End-user can express query entirely in hiragana!

Word Tokenization in Chinese

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.

White space segmentation

Single character segmentation

Subword Tokenization

Subword tokenization

Tokens can be parts of words as well as whole words

Standard baseline segmentation algorithm:

- Maximum Matching (also called Greedy)
  - It follows a greedy approach by attempting to find the longest word match in a given text based on a predefined lexicon or dictionary.
- Byte Pair Encoding (BPE)
  - BPE operates by iteratively merging the most frequent pairs of consecutive characters or character sequences.

Byte Pair Encoding (BPE)

BPE (Byte-Pair Encoding) is a data compression algorithm for text that works by replacing the most frequent pair of characters (bytes) in the input with a single, unused character. This process is repeated until a certain number of symbols is reached.

AAABAAC |V|=3

Pairs: AA,AA,AB, BA, AA, AC

AA occurs thrice so add in vocab, X=AA ; V = {A, B, C, X}

XABXC → XA, AB, BX, XC

BPE Token learner

Let vocabulary be the set of all individual charc
={A, B, C,… a, b, c, …}
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 vocab
Replace every adjacent A, B with AB

Until *K* merges have been done

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

```
V ← 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 ← Most frequent pair of adjacent tokens in C
    t_NEW ← t_L + t_R                         # make new token by concatenating
    V ← 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
```

BPE Token learner

Most subword algorithm are executed inside space-separated tokens.

We commonly first add a special end of word symbol _ before space in training corpus.

Next, separate into letters.

Low, low, low, low, low, lowest, lowest, newer, newer, newer, newer, newer, newer, wider, wider, wider, new, new

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 _	

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 _	

Merge er _ to er_

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 _	

S

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_