Tokenization, Normalization, Stemming

- **corpus (plural corpora)** - computer-readable corpora collection of text or speech.
  - For example the Brown corpus is a million-word collection of samples from 500 written English texts from different genres (newspaper, fiction, non-fiction, academic, etc.), assembled at Brown University in 1963–64 (Kucera and Francis, 1967).

  - **“He stepped out into the hall, was delighted to encounter a water brother”.**
    - This sentence has 13 words if we don’t count punctuation marks as words, 15 if we count punctuation.
    - Whether we treat period (“.”), comma (“,”), and so on as words depends on the task.
    - Punctuation is critical for finding boundaries of things (commas, periods, colons) and for identifying some aspects of meaning
    - part-of-speech tagging or parsing or speech synthesis, punctuations are treated as separate words

- The best way for creating the corpus is to build a datasheet or data statement for each corpus.
- A 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?

    For example, was there a task? 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?

Basic Terminologies

Example: **“do uh main- mainly business data processing”**

- **Fragment** - The broken-off word main- is called a fragment.
- Words like uh and um are called **fillers or filled pauses**. It depends on the application whether to consider this or not

- Example: **Cat and cats**
  - Have same lemma **cat** , but are different **word forms**
  - A **lemma** is a set of lexical forms having the same stem, the same major part-of-speech, and the same word sense.
  - The **wordform** is the full inflected or derived form of the word

- **Types:** are the number of distinct words in a corpus
  - If the set of words in the vocabulary is V, the number of types is the word token vocabulary size |V|.
- **Tokens(Word Tokens)** are the total number of running words in the corpora

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

Rough numbers of types and tokens for some English language corpora

Herdan's Law or Heaps' Law

- Relationship between the number of types |V| and number of tokens N

$$|V| = kN^{\beta}$$

where k and β are positive constants, and $0 < \beta < 1$

- The value of β depends on the corpus size and the genre, but generally ranges from 0.4 to 0.6.

Example:

N (TOTAL NUMBER OF WORD)	V (UNIQUE WORDS)
2100	728
4162	1120
6000	1458
8000	1800
10000	2022

```
V(n) = K n^β  => log V(n) = β log(n) + K

Use, V(n) = 1458 and n = 6000
    => log(1458) = β log(6000) + K     --(1)

    V(n) = 1800 and n = 8000
    => log(1800) = β log(8000) + K     --(2)
```

- Solving (1) and (2) , we get β = 0.69 and K = 3.71

- For example, for the 12000 total words Heaps' law predicts:

  3.71 x 12000^0.69 = **2421** unique words

- The actual number of unique words in 12000 total words is **2340**, which is very close to the predicted result

- Another measure of the number of words in the language is the number of lemmas instead of wordform types.

  (A lemma is a set of lexical forms having the same stem, the same major part- of-speech, and the same word sense)

- Dictionaries can help in giving lemma counts

- Dictionary entries or boldface forms are a very rough upper bound on the number of lemmas

Text Normalization

- Before almost any natural language processing of a text, the text has to be normalized.

- At least three tasks are commonly applied as part of any normalization process:

1. Tokenizing (segmenting) words
2. Normalizing word formats
3. Segmenting sentences

Word Tokenization

- The task of segmenting running text into words
- Often need to break off punctuation as a separate token;
  - commas are a useful piece of information for parsers, periods help indicate sentence boundaries
- But often want to keep the punctuation that occurs word internally, in examples like m.p.h., Ph.D., AT&T, and cap'n.
- Special characters and numbers will need to be kept in prices ($45.55) and dates (01/02/06);
  - don't want to segment that price into separate tokens of "45" and "55".
- And there are URLs (http://www.stanford.edu), Twitter hashtags (#nlproc), or email addresses (someone@cs.colorado.edu).

- A tokenizer can also be used to expand **clitic** contractions that are marked by apostrophes,
  - for example, converting **what're** to the two tokens **what are**

- A **clitic** is a part of a word that can’t stand on its own, and can only occur when it is attached to another word

- Tokenization algorithms may also tokenize multiword expressions like **New York** or **rock 'n' roll** as a single token, which requires a multiword expression dictionary of some sort.
- Tokenization is thus intimately tied up with **named entity recognition**, the task of detecting names, dates, and organizations

- One commonly used tokenization standard is known as the Penn Treebank tokenization standard, used for the parsed corpora (treebanks) released by the Linguistic Data Consortium (LDC), the source of many useful datasets.

- This standard separates out clitics (**doesn't** becomes **does** plus **n't**), keeps hyphenated words together, and separates out all punctuation

- Tokenization needs to be run before any other language processing
- The standard method for tokenization is to use deterministic algorithms based on regular expressions compiled into very efficient finite state automata

- Example of a basic regular expression that can be used to tokenize with the nltk.regexp tokenize function of the Python-based Natural Language Toolkit (NLTK) (Bird et al. 2009; http://www.nltk.org).

```
>>> 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', '...']
```

- Deterministic algorithms can deal with the ambiguities that arise

  Example: Apostrophe needs to be tokenized differently when used as a genitive marker (as in the **book's cover**), a quotative as in **'The other class', she said**, or in clitics like **they're**.

Byte-Pair Encoding for Tokenization

- Instead of defining tokens as words (whether delimited by spaces or more complex algorithms), or as characters, use the data to automatically tell us what the tokens should be.

- Useful in dealing with unknown words, an important problem in language processing

- NLP algorithms often learn some facts about language from one corpus (a **training corpus**) and then use these facts to make decisions about a separate **test corpus** and its language.
  - Thus if our training corpus contains, say the words ***low, new, newer***, but not ***lower***, then if the word ***lower*** appears in our test corpus, our system will not know what to do with it

- Modern tokenizers often automatically induce sets of tokens that include tokens smaller than words, called **subwords**.
- **Subwords** can be arbitrary substrings, or they can be meaning-bearing units like the morphemes ***-est*** or ***-er***. (Morphemes)

- In modern tokenization schemes, most tokens are words, but some tokens are frequently occurring morphemes or other subwords like ***-er***.
- Every unseen words like ***lower*** can thus be represented by some sequence of known subword units, such as ***low*** and ***er***

- Most tokenization schemes have two parts:
  - **a token learner, and a token segmenter**.
- The ***token learner*** takes a raw *training corpus* (sometimes roughly preseparated into words, for example by whitespace) and induces a vocabulary, a set of tokens.
- The ***token segmenter*** takes a raw test sentence and segments it into the tokens in the vocabulary.

- Three algorithms are widely used:
  - **byte-pair encoding** (Sennrich et al., 2016),
  - **unigram language modeling** (Kudo, 2018), and
  - **WordPiece** (Schuster and Nakajima, 2012);
- There is also a **SentencePiece** library that includes implementations of the first two of the three

Byte-pair encoding Algorithm

Step 1: The BPE token learner begins with a vocabulary that is just the set of all individual characters.

Step 2 : It then examines the training corpus, chooses the two symbols that are most frequently adjacent (say 'A', 'B'), adds a new merged symbol 'AB' to the vocabulary, and replaces every adjacent 'A' 'B' in the corpus with the new 'AB'

Step 3: It continues to count and merge, creating new longer and longer character strings, until k merges have been done creating k novel tokens; k is thus a parameter of the algorithm.

The resulting vocabulary consists of the original set of characters plus k new symbols.

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

Example: Input corpus of 18 word tokens with counts for each word (the word low appears 5 times, the word newer 6 times, and so on), which would have a starting vocabulary of 11 letters

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 _	

- The most frequent is the pair **e r** because it occurs in newer (frequency of 6) and wider (frequency of 3) for a total of 9 occurrences.

- Treating ***er*** as one symbol, and counted again:

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 _	

- Now the most frequent pair is ***er_*** , which we merge; system has learned that there should be a token for word-final ***er***, represented as ***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 _	

- Next ***n e*** (total count of 8) get merged 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 _	

- If continued, 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_

- Once vocabulary is learned, the ***token parser*** is used to tokenize a test sentence. The token parser just runs on the test data the merges that has been have learned from the training data, greedily,

- First segment each test sentence or word into characters.
- Then apply the first rule: replace every instance of ***e r*** in the test corpus with e**r**
- The apply the second rule: replace every instance of ***er*** in the test corpus with ***er_*** , and so on.
- By the end, if the test corpus contained the word ***n e w e r_*** , it would be tokenized as a full word.

- In real algorithms BPE is run with many thousands of merges on a very large input corpus.

- After training :
  - On the test data run each merge learned on training data.
  - Test set “n e w e r _” will be tokenized as a full word
  - Test set “l o w e r _” will be tokenized as “low er_”

Word Normalization, Lemmatization and Stemming

- Word normalization is the task of putting words/tokens in a standard format, choosing a single normal form for words with multiple forms Example: ***USA*** and ***US*** or ***uh-huh*** and ***uhhuh***.

- **Disadvantage**: The spelling information is lost in the normalization process. (but still useful in many applications)

Case folding

- Case folding is a kind of normalization.
- Mapping everything to lower case means that ***Woodchuck*** and ***woodchuck*** are represented identically, which is very helpful for generalization in many tasks, such as information retrieval or speech recognition.

- For sentiment analysis and other text classification tasks, information extraction and machine translation, by contrast, case can be helpful and case folding is generally not done.
  - This is because maintaining the difference between, for example, ***US*** the country and ***us*** the pronoun can outweigh the advantage in generalization that case folding would have provided for other words.

Example text normalization

- Input: “The quick BROWN Fox Jumps OVER the lazy dog.”
- Output: “the quick brown fox jumps over the lazy dog.”

Advantages

- It eliminates case sensitivity, making text data consistent and easier to process.
- It reduces the dimensionality of the data, which can improve the performance of NLP algorithms.

Disadvantages

- It can lead to loss of information, as capitalization can indicate proper nouns or emphasis.

Punctuation Removal

- Punctuation removal is the process of removing special characters and punctuation marks from the text. This technique is useful when working with text data containing many punctuation marks, which can make the text harder to process.

Example text normalization

- Input: “The quick BROWN Fox Jumps OVER the lazy dog!!!”
- Output: “The quick BROWN Fox Jumps OVER the lazy dog”

Advantages

- It removes unnecessary characters, making the text cleaner and easier to process.
- It reduces the dimensionality of the data, which can improve the performance of NLP algorithms.

Disadvantages

- It can lead to loss of information, as punctuation marks can indicate sentiment or emphasis.

Stop Word Removal

- Stop word removal is the process of removing common words with little meaning, such as “the” and “a”. This technique is useful when working with text data containing many stop words, which can make the text harder to process.

Example text normalization

- Input: “The quick BROWN Fox Jumps OVER the lazy dog.”
- Output: “quick BROWN Fox Jumps OVER lazy dog.”

Advantages

- It removes unnecessary words, making the text cleaner and easier to process.
- It reduces the dimensionality of the data, which can improve the performance of NLP algorithms.

Disadvantages

- It can lead to loss of information, as stop words can indicate context or sentiment.

Removing numbers and symbol to normalize the text in NLP

- This technique is useful when working with text data that contain numbers and symbols that are not important for the NLP task.

Example text normalization

- Input: “I have 2 apples and 1 orange #fruits”
- Output: “I have apples and orange fruits”

Advantages

- It removes unnecessary numbers and symbols, making the text cleaner and easier to process.
- It reduces the dimensionality of the data, which can improve the performance of NLP algorithms.

Disadvantages

- It can lead to loss of information, as numbers and symbols can indicate quantities or sentiments.

Removing any remaining non-textual elements to normalize the text in NLP

- Removing any remaining non-textual elements such as HTML tags, URLs, and email addresses This technique is useful when working with text data that contains non-textual elements such as HTML tags, URLs, and email addresses that are not important for the NLP task.

  Example text normalization

  - Input: “Please visit example.com for more information or contact me at info@example.com”
  - Output: “Please visit for more information or contact me at “

- **Advantages**
- It removes unnecessary non-textual elements, making the text cleaner and easier to process.
- It reduces the dimensionality of the data, which can improve the performance of NLP algorithms.
- **Disadvantages**
- It can lead to loss of information, as non-textual elements can indicate context or sentiment.

- Many natural language processing situations one may need two morphologically different forms of a word to behave similarly.
    - For example in web search, someone may type the string ***woodchucks*** but a useful system might want to also return pages that mention ***woodchuck*** with no ***s***.

Lemmatization

- Lemmatization is the task of determining that two words have the same root, despite their surface differences.
- Lemmatization reduces words to their base form by considering the context in which they are used, such as “running” will become “run”.
- The words **dinner** and **dinners** both have the **lemma dinner**

- 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.**
  - 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.

Example text lemmatization

- Input: “running,runner,ran”
- Output: “run, run, run”

Advantages

- It reduces the dimensionality of the data, which can improve the performance of NLP algorithms.
- It makes it easier to identify the core meaning of a word while preserving context.

Disadvantages

- It can be more computationally expensive.
- It may not be able to handle all words or forms.

Stemming

- Stemming reduces the words to their root form by removing suffixes and prefixes, such as “running” will be “run”. This method is helpful when working with text data that has many different versions of the same word, which can make the text harder to process.

  Example text normalization

  - Input: “running, runner, ran, caress”
  - Output: “run, run, run, car”

Advantages

- It reduces the dimensionality of the data, which can improve the performance of NLP algorithms.
- It makes it easier to identify the core meaning of a word.

Disadvantages

- It can lead to loss of information, as the root form of a word may not always be the correct form.
- It may produce non-existent words

Stemming	Lemmatization
Stemming is a process that stems or removes few characters from a word, often leading to incorrect meanings and spelling.	**Lemmatization** considers the context and converts the word to its meaningful base form, which is called Lemma.
For instance, stemming the word '**Caring**' would return '**Car**'.	For instance, lemmatizing the word '**Caring**' would return '**Care**'.
Stemming is used in case of large dataset where performance is an issue.	Lemmatization is computationally expensive since it involves look-up tables and what not.

The Porter Stemmer

- The naive version of morphological analysis is called stemming
- widely used stemming algorithms is the Porter (1980).

Basic Terminologies

- A **consonant** in a word is a letter other than A, E, I, O or U, and other than Y preceded by a consonant. (The fact that the term **consonant** is defined to some extent in terms of itself does not make it ambiguous.)
- Example: TOY the consonants are T and Y, and in SYZYGY they are S, Z and G. If a letter is not a consonant it is a vowel.
- The consonants are represented by ‘c’ and vowels are represented by ‘v’.

- A list ccc... of length greater than 0 will be denoted by C, and a list vvv... of length greater than 0 will be denoted by V.
- Any word, or part of a word, therefore has one of the four forms:
  - **CVCV … C** → collection, management
  - **CVCV … V** → conclude, revise
  - **VCVC … C** → entertainment, illumination
  - **VCVC … V** → illustrate, abundance

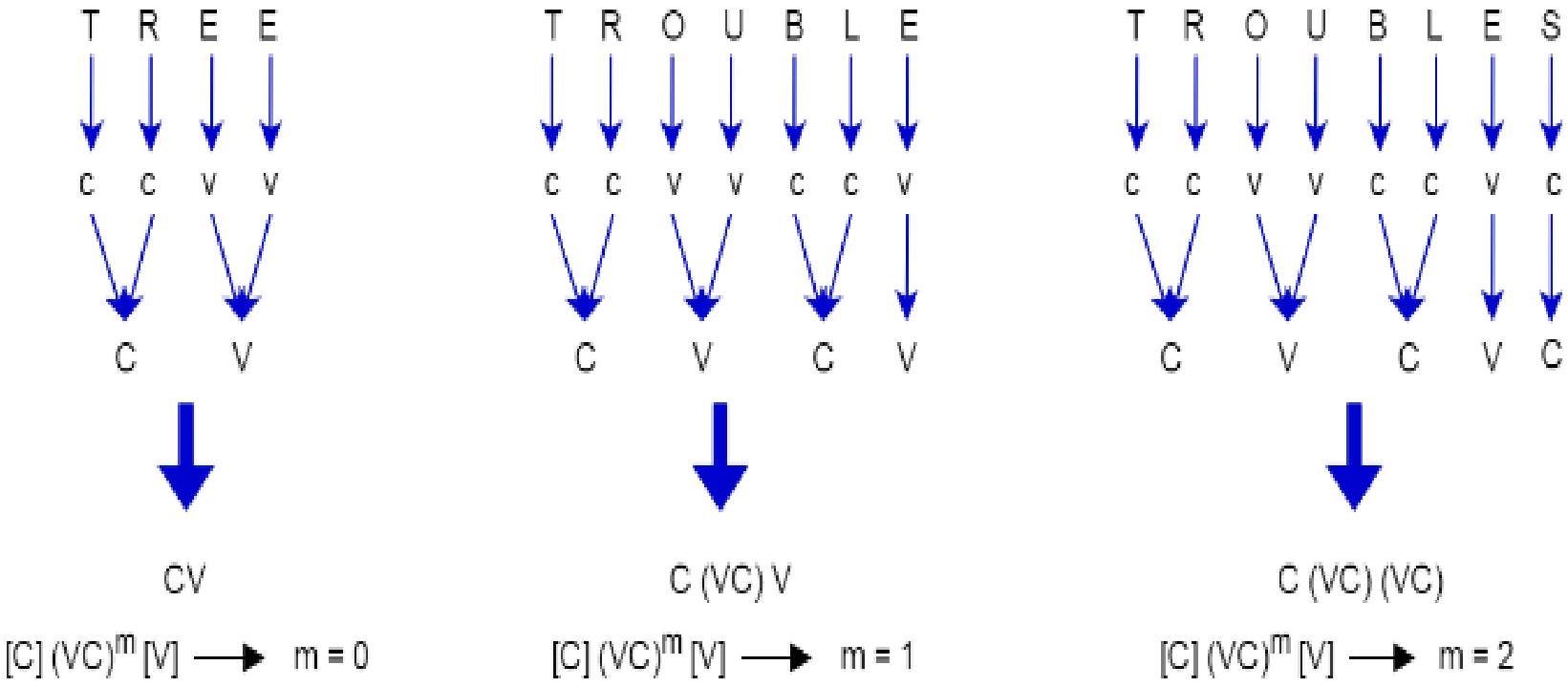
- These may all be represented by the single form

  [C]VCVC ... [V]

  where the square brackets denote arbitrary presence of their contents.
- Using (VC^m) to denote VC repeated m times, this may again be written as

  [C]($ VC^m $)[V]
- m will be called the measure of any word or word part when represented in this form.

  The case m = 0 covers the null word.



- Examples:
  - m=0 TR, EE, TREE, Y, BY.
  - m=1 TROUBLE, OATS, TREES, IVY.
  - m=2 TROUBLES, PRIVATE, OATEN, ORRERY.

- The rules for removing a suffix will be given in the form

  (condition) S1 -> S2

- This means that if a word ends with the suffix **S1**, and the stem before **S1** satisfies the given condition, **S1** is replaced by **S2**. The condition is usually given in terms of **m**,

  Example: **(m > 1) EMENT ->**

  Here **S1** is '**EMENT**' and **S2** is **null**. This would map **REPLACEMENT** to **REPLAC**, since **REPLAC** is a word part for which **m = 2**.

- The conditions may contain the following:
  - *S – the stem ends with S (and similarly for the other letters)
  - *v* – the stem contains a vowel
  - m=2 - TROUBLES, PRIVATE, OATEN, ORRERY.
  - *d – the stem ends with a double consonant (e.g. -TT, -SS)
  - *o – the stem ends cvc, where the second c is not W, X or Y (e.g. -WIL, -HOP)

- And the condition part may also contain expressions with and, or and not, so that:

`(m>1 and (*S or *T))` : tests for a stem with m>1 ending in S or T, while

`(*d and not (*L or *S or *Z))` : tests for a stem ending witha double consonant other than L, S or Z.

- In a set of rules written beneath each other, only one is obeyed, and this will be the one with the longest matching **S1** for the given word.
- For example, with the following rules,
  - ISSES -> SS (Example : caresses -> caress)
  - IES -> I (Example : ties -> ti)
  - SS -> SS (Example : caress -> caress)
  - S -> (Example : cats -> cat)

  - The conditions are all null.
  - CARESSES maps to CARESS since SSES is the longest match for S1.
  - CARESS maps to CARESS (since S1=”SS”) and
  - CARES to CARE (since S1=”S”).

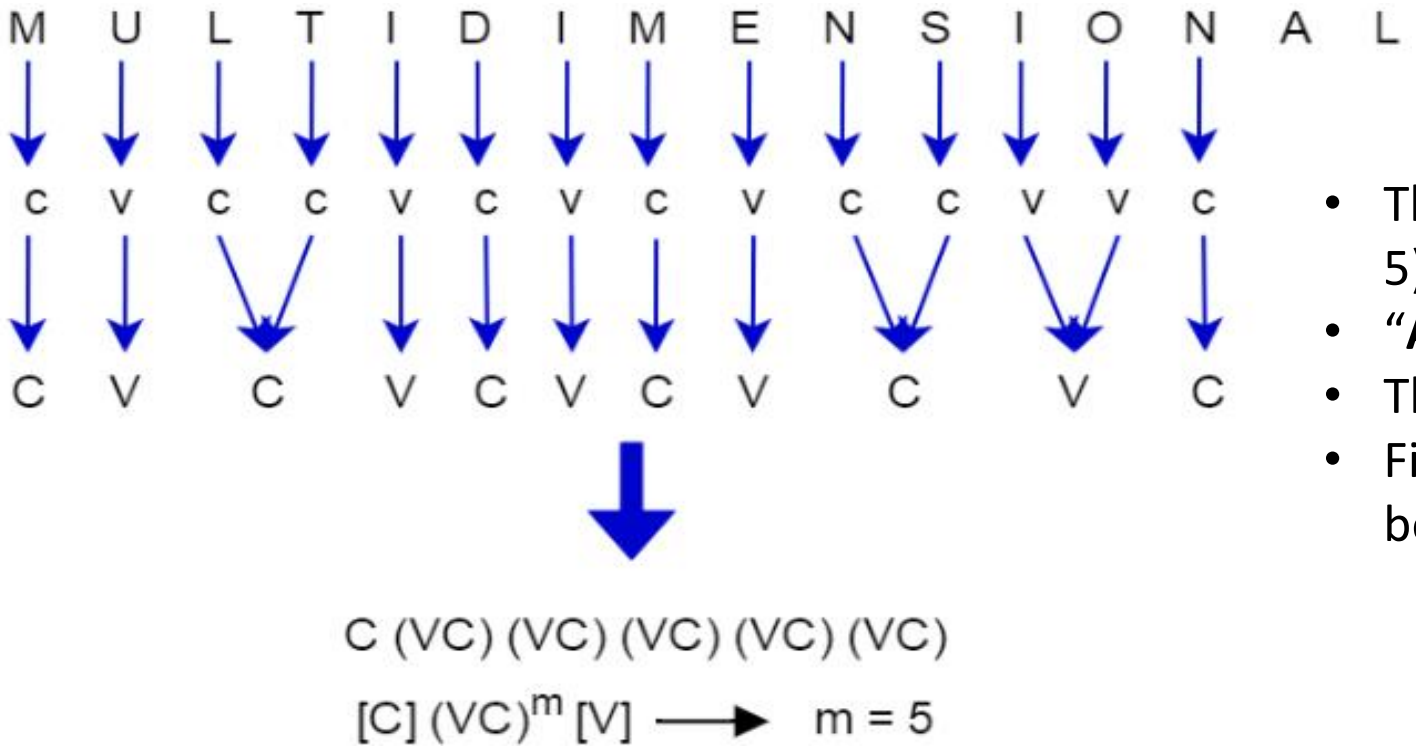
- (m>0) EED -> EE (Example : feed -> feed ; agreed -> agree)
- (*v*) ED -> (Example : plastered -> plaster ; bled -> bled)
- (*v*) ING -> (Example : motoring -> motor ; sing -> sing)
- S -> (Example : cats -> cat)

- T -> ATE (Example : conflat(ed) -> conflate)
- BL -> BLE (Example : troubl(ed) -> trouble)
- IZ -> IZE (Example : siz(ed) -> size)

- (m>0) ATIONAL -> ATE (Example : relational -> relate)
- (m>0) TIONAL -> TION (Example : conditional -> condition ; rational -> rational)
- (m>0) ENCI -> ENCE (Example : valenci -> valence)
- Etc,….

Example 1:

Input word : **MULTIDIMENSIONAL**

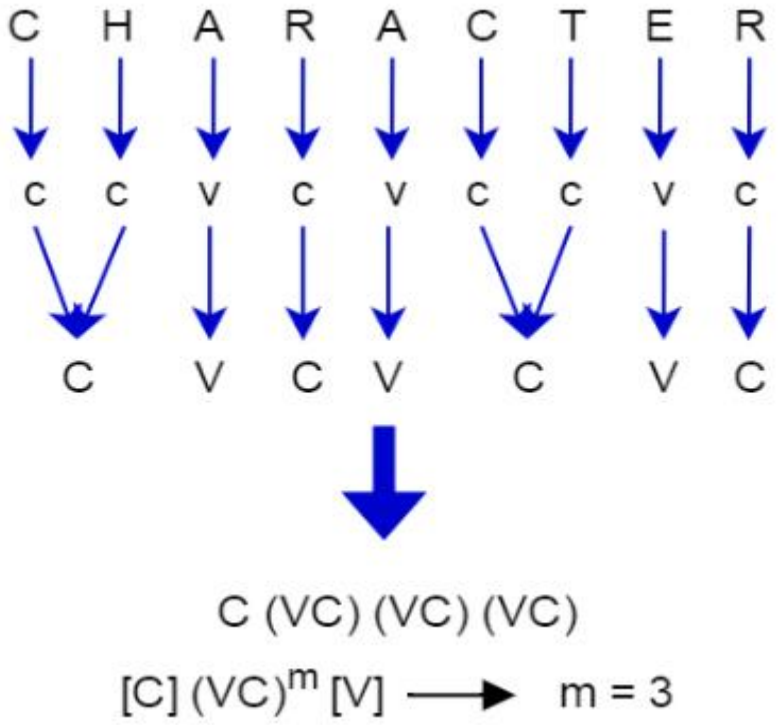


- The stem of the word has m > 1 (since m = 5) and ends with "**AL**".
- "**AL**" is deleted (replaced with null).
- The word will not change the stem further.
- Finally the output will be **MULTIDIMENSION**.

(m>1) MULTIDIMENSIONAL → **MULTIDIMENSION**

Example 2

Input word : **CHARACTERIZATION**



- The stem of the word has m > 0 (since m = 3) and ends with "**IZATION**".
- "**IZATION**" will be replaced with "**IZE**". Coz: ***((m>0) IZATION -> IZE)***
- Then the new stem will be **CHARACTERIZE**.
- Now m > 1 (since m = 3) and the stem ends with **"IZE"**.
- **"IZE"** will be deleted (replaced with null). ***(m>1) IZE -> Null)***
- No change will happen to the stem in other steps.
- Finally the output will be **CHARACTER**.

CHARACTERIZATION → CHARACTERIZE → **CHARACTER**

END