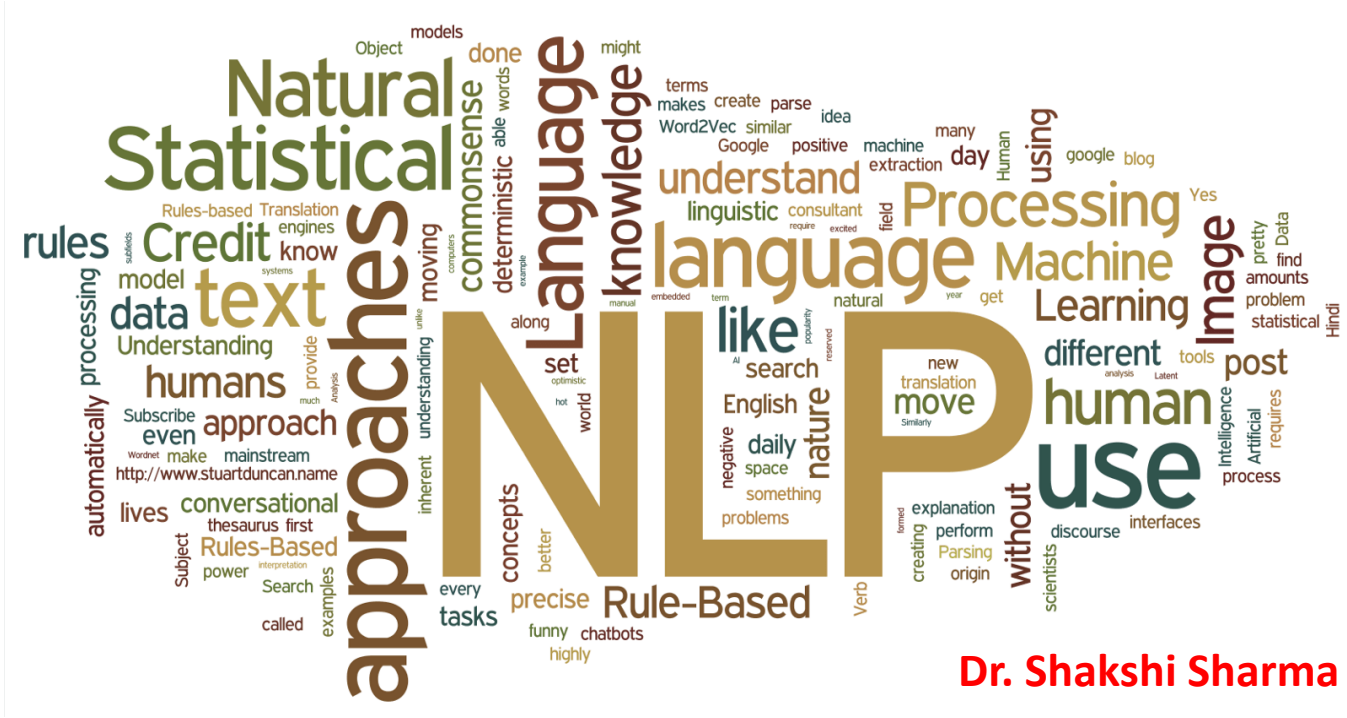
Natural Language Processing



Dr. Shakshi Sharma

Assistant Professor

School of Artificial Intelligence (SoAI), Bennett University

PhD|Junior Researcher|Estonia (Europe)

Research Intern|University of Sheffield, United Kingdom (UK)

Website: https://sites.google.com/view/shakshi-sharma/home

Stemming

Stemming

- Stemming is a mechanism that **eliminates prefixes and suffixes** from words, transforming them into their **fundamental or root form**.
- The stemming technique enhance the effectiveness of the text processing tasks.

Example: The words "**programming**," "**programmer**," and "**programs**" stemmed into the common word stem "**program**."

- Stemming in natural language processing reduces words to their base or root form, aiding in text normalization for easier processing.
- This is an important steps in text processing tasks like text classification, information retrieval, and text summarization.
- While beneficial, stemming has **drawbacks**, including potential impacts on text readability and occasional inaccuracies in determining the correct root form of a word.

Stemming

- Some more example of stemming for root word "like" include:

->"likes"

->"liked"

->"likely"

->"liking"

Types of Stemmer in NLTK

Python NLTK contains a variety of stemming algorithms, including several types. Let’s examine them down below.

1. **Porter’s Stemmer**
2. **Lovins Stemmer**
3. **Krovetz Stemmer**
4. **Xerox Stemmer**
5. **N-Gram Stemmer**
6. **Snowball Stemmer**
7. **Lancaster Stemmer**

Porter's Stemmer

It is one of the most popular stemming methods proposed in 1980.

- It is based on the **idea** that the suffixes in the English language are made up of a combination of smaller and simpler suffixes.
- This stemmer is known for its **speed** and **simplicity**.
- The Porter Stemmer widely used in data mining and Information retrieval.
- However, its applications are only **limited** to English words. Also, the group of stems is mapped on to the same stem and the output stem is not necessarily a meaningful word**.**

Example: EED -> EE means "if the word has at least one vowel and consonant plus EED ending, change the ending to EE" as 'agreed' becomes 'agree'.

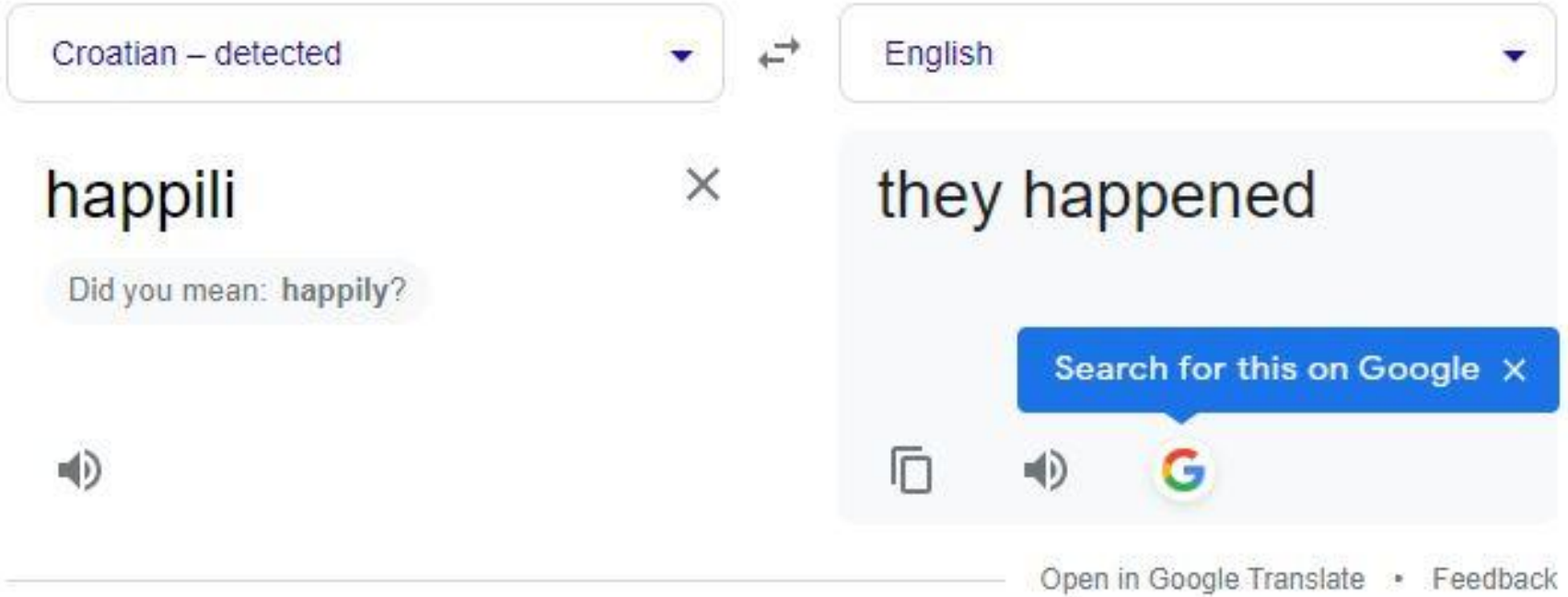
Porter's Stemmer

```
from nltk.stem import PorterStemmer
# Create a Porter Stemmer instance
porter_stemmer = PorterStemmer()
# Example words for stemming
words = ["running", "jumps", "happily", "running", "happily"]
# Apply stemming to each word
stemmed_words = [porter_stemmer.stem(word) for word in words]
# Print the results
print("Original words:", words)
print("Stemmed words:", stemmed_words)
```

Output

```
Original words: ['running', 'jumps', 'happily', 'running', 'happily']
Stemmed words: ['run', 'jump', 'happili', 'run', 'happili']
```

Porter's Stemmer



- **Advantage**: It produces the best output as compared to other stemmers and it has less error rate.
- **Limitation**: Morphological variants produced are not always real words.

Lovins Stemmer

- It is proposed by Lovins in 1968, that **removes the longest suffix from a word** then the word is recorded to convert this stem into valid words.

Example: sitting -> sitt -> sit

- **Advantage**: It is fast and handles irregular plurals like ‘teeth’ and ‘tooth’ etc.
- **Limitation**: It is time consuming and frequently fails to form words from stem.

Krovetz Stemmer

It was proposed in 1993 by Robert Krovetz. Following are the steps:

- Convert the plural form of a word to its singular form.
- Convert the past tense of a word to its present tense and remove the suffix ‘ing’.

Example: ‘children’ -> ‘child’

- **Advantage**: It is light in nature and can be used as pre-stemmer for other stemmers.
- **Limitation**: It is inefficient in case of large documents.

Xerox Stemmer

- Capable of processing extensive datasets and generating valid words
- It is language-dependent

Example:

‘children’ -> ‘child’

‘understood’ -> ‘understand’

‘whom’ -> ‘who’

‘best’ -> ‘good’

N-Gram Stemmer

- The algorithm, apply named n-grams (typically n=2 or 3), involves breaking words into segments of length n and then applying statistical analysis to identify patterns. An n-gram is a set of n consecutive characters extracted from a word in which similar words will have a high proportion of n-grams in common.

- Example: ‘INTRODUCTIONS’ for n=2 becomes : *I, IN, NT, TR, RO, OD, DU, UC, CT, TI, IO, ON, NS, S*

- **Advantage**: It is based on string comparisons and it is language independent.
- **Limitation**: It requires space to create and index the n-grams and it is not time efficient.

Snowball Stemmer

- The Snowball Stemmer, compared to the Porter Stemmer, is **multi-lingual** as it can handle non-English words.
- It supports various languages and is based on the ‘Snowball’ programming language, known for efficient processing of small strings.
- The Snowball stemmer is way more aggressive than Porter Stemmer and is also referred to as **Porter2 Stemmer**.
- Because of the improvements added when compared to the Porter Stemmer, the Snowball stemmer is having **greater computational speed**.

Snowball Stemmer

```
from nltk.stem import SnowballStemmer
# Choose a language for stemming, for example, English
stemmer = SnowballStemmer(language='english')
# Example words to stem
words_to_stem = ['running', 'jumped', 'happily', 'quickly', 'foxes']
# Apply Snowball Stemmer
stemmed_words = [stemmer.stem(word) for word in words_to_stem]
# Print the results
print("Original words:", words_to_stem)
print("Stemmed words:", stemmed_words)
```

Output:

```
Original words: ['running', 'jumped', 'happily', 'quickly', 'foxes']
Stemmed words: ['run', 'jump', 'happili', 'quick', 'fox']
```

Lancaster Stemmer

- The Lancaster stemmers are more aggressive and dynamic compared to the other two stemmers.
- The stemmer is really faster, but the algorithm is really confusing when dealing with small words.

Lancaster Stemmer

```
from nltk.stem import LancasterStemmer
# Create a Lancaster Stemmer instance
stemmer = LancasterStemmer()
# Example words to stem
words_to_stem = ['running', 'jumped', 'happily', 'quickly', 'foxes']
# Apply Lancaster Stemmer
stemmed_words = [stemmer.stem(word) for word in words_to_stem]
# Print the results
print("Original words:", words_to_stem)
print("Stemmed words:", stemmed_words)
```

Output:

```
Original words: ['running', 'jumped', 'happily', 'quickly', 'foxes']
Stemmed words: ['run', 'jump', 'happy', 'quick', 'fox']
```

Regexp Stemmer

- The Regexp Stemmer, or Regular Expression Stemmer, is a stemming algorithm that utilizes regular expressions to identify and remove suffixes from words.

- It allows users to **define custom rules** for stemming by specifying patterns to match and remove.

- This method provides flexibility and control over the stemming process, making it suitable for specific applications where custom rule-based stemming is desired.

Regexp Stemmer

```
from nltk.stem import RegexpStemmer
# Create a Regexp Stemmer with a custom rule
custom_rule = r'ing$'
regexp_stemmer = RegexpStemmer(custom_rule)
# Apply the stemmer to a word
word = 'running'
stemmed_word = regexp_stemmer.stem(word)
print(f'Original Word: {word}')
print(f'Stemmed Word: {stemmed_word}’)
```

Output:

```
Original Word: running
Stemmed Word: runn
```

Applications of Stemming

- **Sentiment Analysis**, which examines reviews and comments made by different users about anything, is frequently used for product analysis, such as for online retail stores. Before it is interpreted, stemming is accepted in the form of the text-preparation mean.

- A method of group analysis used on textual materials is called **document clustering** (also known as text clustering). Important uses of it include subject extraction, automatic document structuring, and quick information retrieval.

Etc.

Disadvantages in Stemming

Over-stemming:

- Over-stemming in natural language processing occurs when a stemmer produces incorrect root forms or non-valid words.
- This can result in a loss of meaning and readability. For instance, "**arguing**" may be stemmed to "**argu**," losing meaning.
- An example of over-stemming is the Lancaster stemmer's reduction of **wander to wand**, two semantically distinct terms in English.
- To address this, choosing an appropriate stemmer, testing on sample text, or using **lemmatization** can mitigate over-stemming issues.
- Techniques like semantic role labeling and sentiment analysis can enhance context awareness in stemming.

Disadvantages in Stemming

Under-stemming:

- Under-stemming in natural language processing arises when a stemmer fails to produce accurate root forms or reduce words to their base form.
- This can result in a loss of information and hinder text analysis.
- An example of under-stemming is the **Porter** stemmer's non-reduction of **knavish to knavish** and **knave to knave**, which do share the same semantic root. By comparison, the **Lovins** stemmer reduces both words to **knav**.