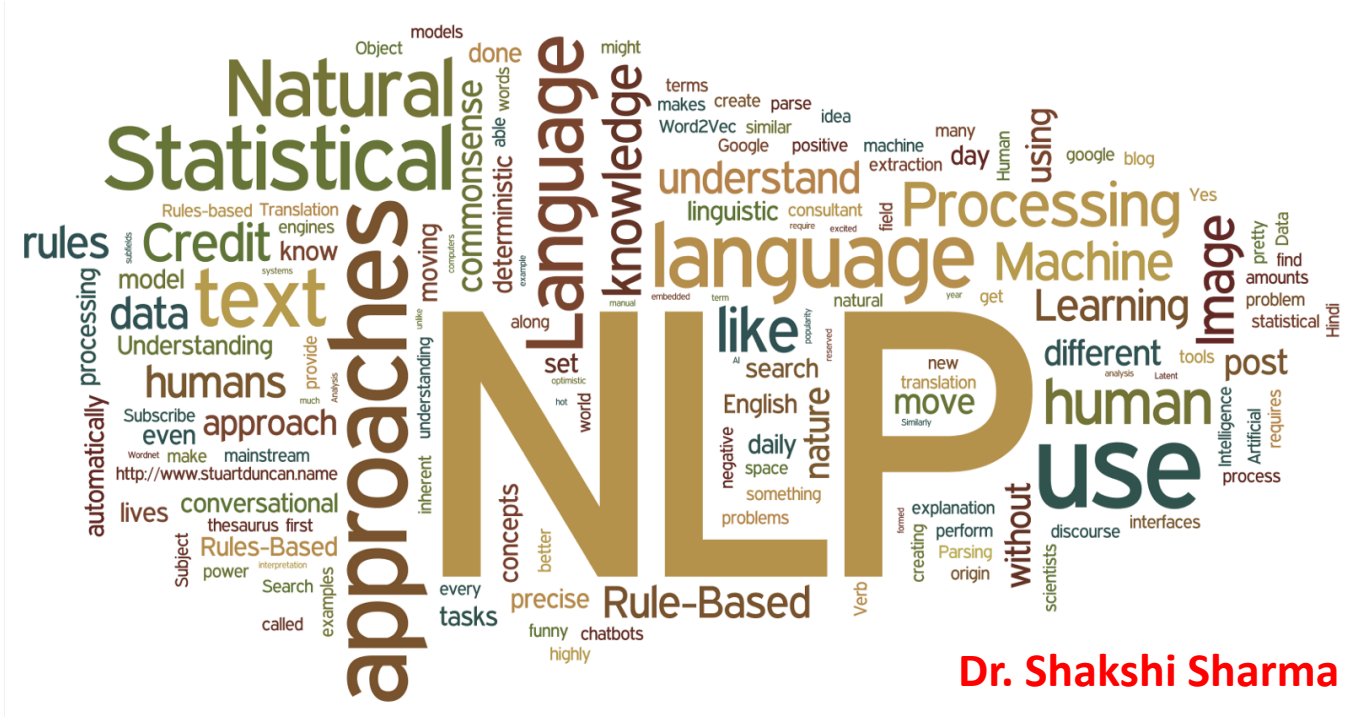
Natural Language Processing



Dr. Shakshi Sharma

Assistant Professor

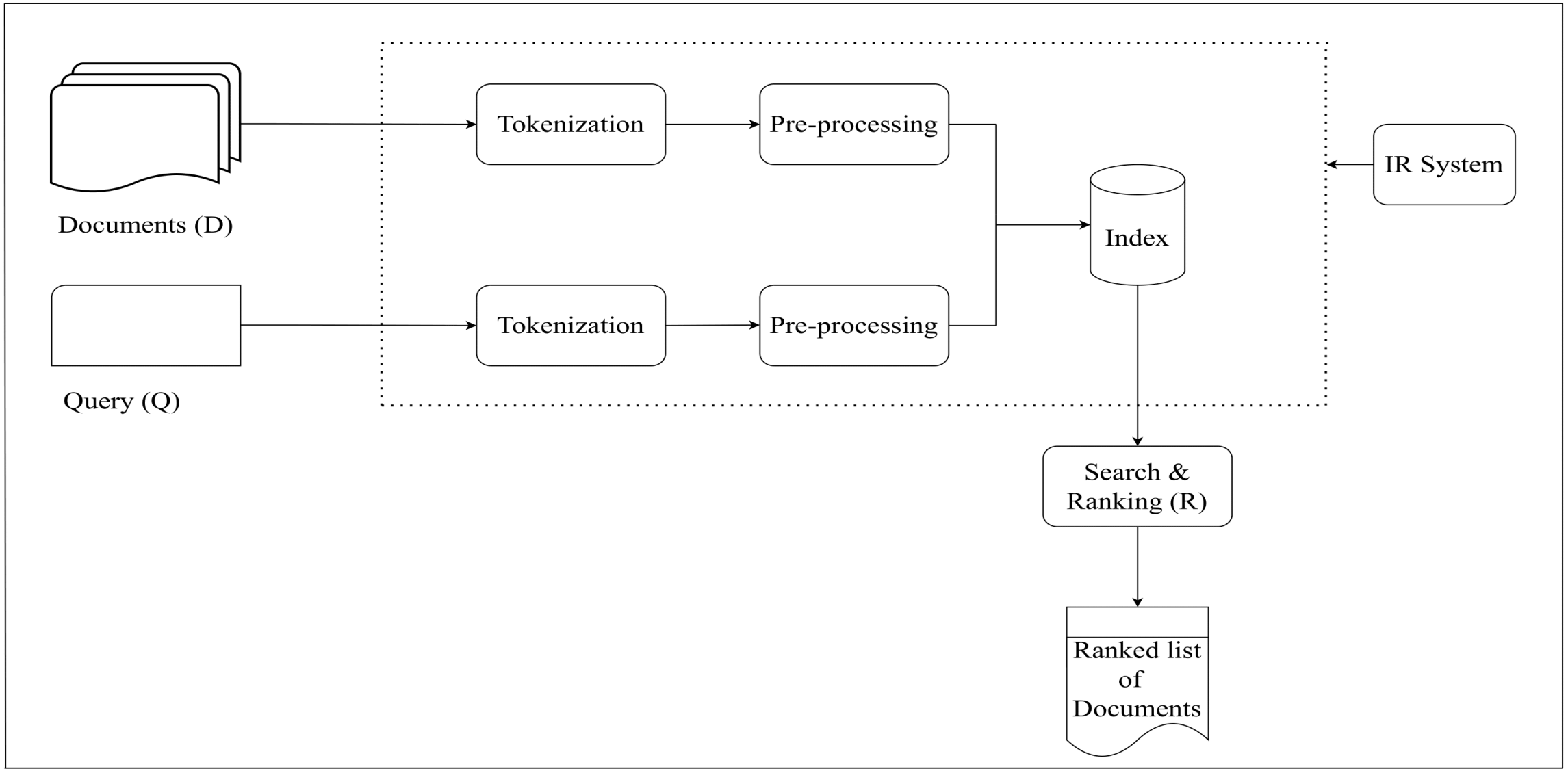
School of Artificial Intelligence (SoAI), Bennett University

PhD|Junior Researcher|Estonia (Europe)

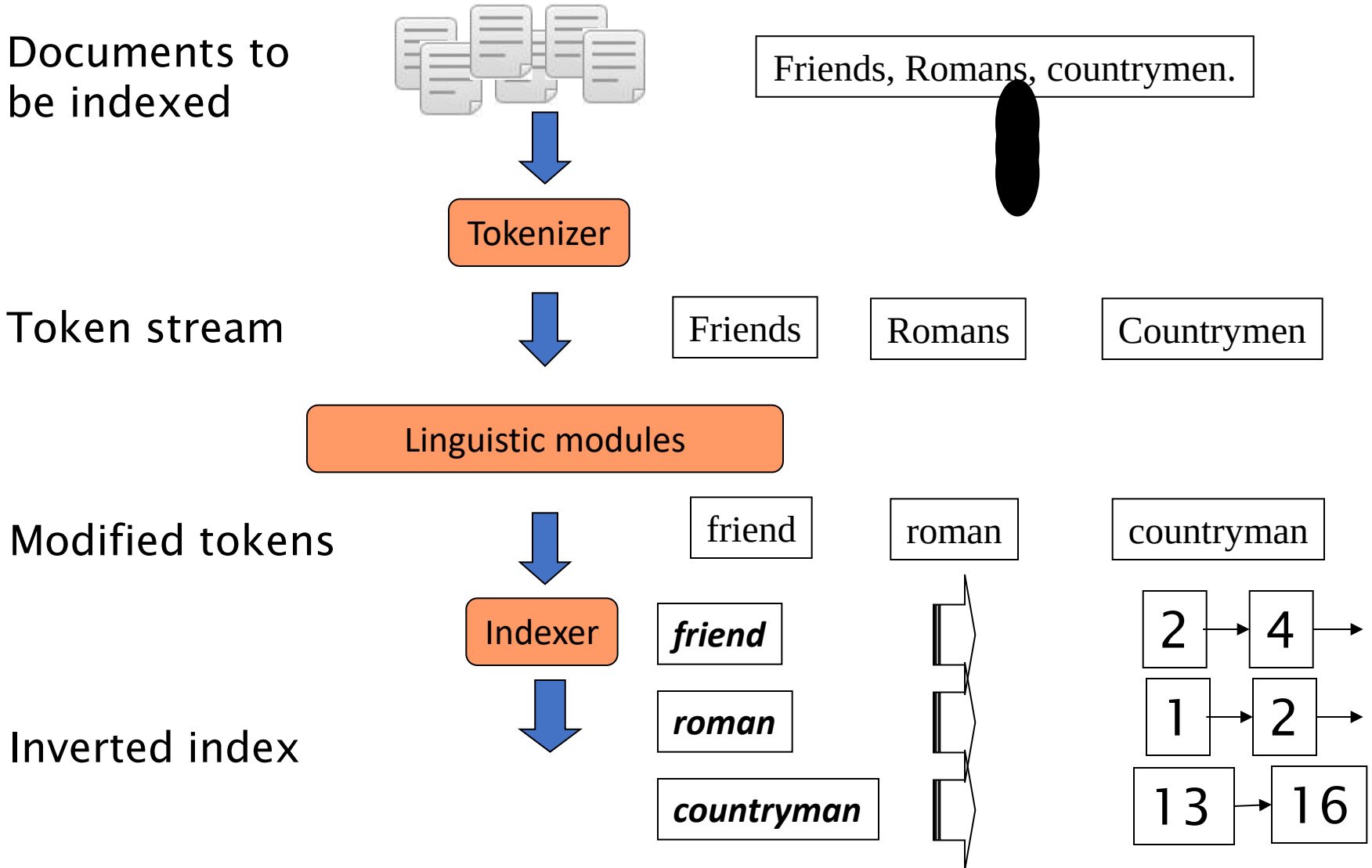
Research Intern|University of Sheffield, United Kingdom (UK)

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

Block diagram of an Information Retrieval Process



Inverted Index Construction



Initial stages of text processing

- Tokenization
  - Cut character sequence into word tokens
- Normalization
  - Map text and query term to same form
    - You want ***U.S.A.*** and ***USA*** to match
- Stemming
  - We may wish different forms of a root to match
    - ***authorize***, ***authorization***
- Stop words
  - We may omit very common words
    - ***the, a, to, of***

Tokenization

Tokenization

- Tokenization is a technique that divides a sentence or phrase into smaller units known as tokens.
- These tokens can encompass words, dates, punctuation marks, or even fragments of words.
- Tokenization is a critical step in NLP tasks such as text processing, language modelling, machine translation etc.

Types of Tokenization

- Tokenization can be classified into several types based on how the text is segmented. Here are some types of tokenization:

I. Word Tokenization:

- Word tokenization divides the text into individual words. Many NLP tasks use this approach, in which words are treated as the basic units of meaning.

- **Example:**

  Input: "Tokenization is an important NLP task."

  Output: ["Tokenization", "is", "an", "important", "NLP", "task", "."]

Types of Tokenization

II. Sentence Tokenization:

- The text is segmented into sentences during sentence tokenization. This is useful for tasks requiring individual sentence analysis or processing.

Example:

Input: "Tokenization is an important NLP task. It helps break down text into smaller units."

Output: ["Tokenization is an important NLP task.", "It helps break down text into smaller units."]

Types of Tokenization

III. Sub word Tokenization:

- Sub word tokenization entails breaking down words into smaller units, which can be especially useful when dealing with morphologically rich languages or rare words.

Example:

Input: "tokenization"

Output: ["token", "ization"]

Types of Tokenization

IV. Character Tokenization:

- This process divides the text into individual characters. This can be useful for modelling character-level language.

Example:

Input: "Tokenization"

Output: ["T", "o", "k", "e", "n", "i", "z", "a", "t", "i", "o", "n"]

Need of Tokenization

- **Effective Text Processing:** Tokenization reduces the size of raw text so that it can be handled more easily for processing and analysis.
- **Feature extraction:** Text data can be represented numerically for algorithmic comprehension by using tokens as features in machine learning models.
- **Language Modelling:** Tokenization in NLP facilitates the creation of organized representations of language, which is useful for tasks like text generation and language modelling.
- **Information Retrieval:** Tokenization is essential for indexing and searching in systems that store and retrieve information efficiently based on words or phrases.
- **Text Analysis:** Tokenization is used in many NLP tasks, including sentiment analysis and named entity recognition, to determine the function and context of individual words in a sentence.

Implementation for Tokenization

The code uses sent_tokenize function from NLTK library. The sent_tokenize function is used to segment a given text into a list of sentences.

```
from nltk.tokenize import sent_tokenize
text = "Hello everyone. Welcome to the Class. You are studying NLP ."
sent_tokenize(text)
```

Output:

```
['Hello everyone.',
 'Welcome to the Class.',
 'You are studying NLP.']
```

Implementation for Tokenization

When we have huge chunks of data then it is efficient to use ‘PunktSentenceTokenizer' from the NLTK library.

```
import nltk.data
# Loading PunktSentenceTokenizer using English pickle file
tokenizer = nltk.data.load('tokenizers/punkt/PY3/english.pickle')
tokenizer.tokenize(text)
```

Tokenize sentence of different language

```
import nltk.data
spanish_tokenizer = nltk.data.load('tokenizers/punkt/PY3/spanish.pickle')
text = 'Hola amigo. Estoy bien.'
spanish_tokenizer.tokenize(text)
```

Output: ??

Implementation for Tokenization

When we have huge chunks of data then it is efficient to use ‘PunktSentenceTokenizer' from the NLTK library.

```
import nltk.data
# Loading PunktSentenceTokenizer using English pickle file
tokenizer = nltk.data.load('tokenizers/punkt/PY3/english.pickle')
tokenizer.tokenize(text)
```

Tokenize sentence of different language

```
import nltk.data
spanish_tokenizer = nltk.data.load('tokenizers/punkt/PY3/spanish.pickle')
text = 'Hola amigo. Estoy bien.'
spanish_tokenizer.tokenize(text)
```

Output:

```
['Hola amigo.',
 'Estoy bien.']
```

Implementation for Tokenization

Word Tokenization using work_tokenize

- The word_tokenize function is helpful for breaking down a sentence or text into its constituent words, facilitating further analysis or processing at the word level in natural language processing tasks.

```
from nltk.tokenize import word_tokenize
text = "Hello, everyone. Welcome to Class."
word_tokenize(text)
```

Output: ??

Implementation for Tokenization

Word Tokenization using work_tokenize

- The word_tokenize function is helpful for breaking down a sentence or text into its constituent words, facilitating further analysis or processing at the word level in natural language processing tasks.

```
from nltk.tokenize import word_tokenize
text = "Hello everyone. Welcome to Class."
word_tokenize(text)
```

Output:

['Hello', 'everyone', '.', 'Welcome', 'to', ‘Class', ‘.’]

Implementation for Tokenization

➢ **Word Tokenization Using TreebankWordTokenizer**

- These tokenizers work by separating the words using punctuation and spaces.
- It doesn’t discard the punctuation, allowing a user to decide what to do with the punctuations at the time of pre-processing.

```
from nltk.tokenize import TreebankWordTokenizer
text = "Hello everyone. Welcome to Class."
tokenizer = TreebankWordTokenizer()
tokenizer.tokenize(text)
```

Output: ??

Implementation for Tokenization

➢ **Word Tokenization Using TreebankWordTokenizer**

- These tokenizers work by separating the words using punctuation and spaces.
- It doesn’t discard the punctuation, allowing a user to decide what to do with the punctuations at the time of pre-processing.

```
from nltk.tokenize import TreebankWordTokenizer
text = "Hello everyone. Welcome to Class."
tokenizer = TreebankWordTokenizer()
tokenizer.tokenize(text)
```

Output:

['Hello', 'everyone.', 'Welcome', 'to', ‘Class', ‘.’]

Implementation for Tokenization

Word Tokenization using WordPunctTokenizer

- The WordPunctTokenizer is one of the NLTK tokenizers that splits words based on punctuation boundaries.
- Each punctuation mark is treated as a separate token.

```
from nltk.tokenize import WordPunctTokenizer
tokenizer = WordPunctTokenizer()
tokenizer.tokenize("Let's see how it's working.")
```

Output: ??

Implementation for Tokenization

Word Tokenization using WordPunctTokenizer

- The WordPunctTokenizer is one of the NLTK tokenizers that splits words based on punctuation boundaries.
- Each punctuation mark is treated as a separate token.

```
from nltk.tokenize import WordPunctTokenizer
tokenizer = WordPunctTokenizer()
tokenizer.tokenize("Let's see how it's working.")
```

Output:

```
['Let', "'", 's', 'see', 'how', 'it', "'", 's', 'working', '.']
```

Implementation for Tokenization

- Using regular expressions allows for more fine-grained control over tokenization, and you can customize the pattern based on your specific requirements.

```
from nltk.tokenize import RegexpTokenizer
tokenizer = RegexpTokenizer(r'\w+')
text = "Let's see how it's working."
tokenizer.tokenize(text)
```

Output: ??

Implementation for Tokenization

- Using regular expressions allows for more fine-grained control over tokenization, and you can customize the pattern based on your specific requirements.

```
from nltk.tokenize import RegexpTokenizer
tokenizer = RegexpTokenizer(r'\w+')
text = "Let's see how it's working."
tokenizer.tokenize(text)
```

Output:

['Let', 's', 'see', 'how', 'it', 's', 'working']

More Techniques for Tokenization

We can also implement tokenization using following methods and libraries:

- Spacy
- BERT tokenizer
- Byte-Pair Encoding
- Sentence Piece

Links for further information:

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

Summary of Byte pair encoding, sentence piece and word piece tokenizers: https://huggingface.co/docs/transformers/en/tokenizer_summary

Limitations of Tokenization

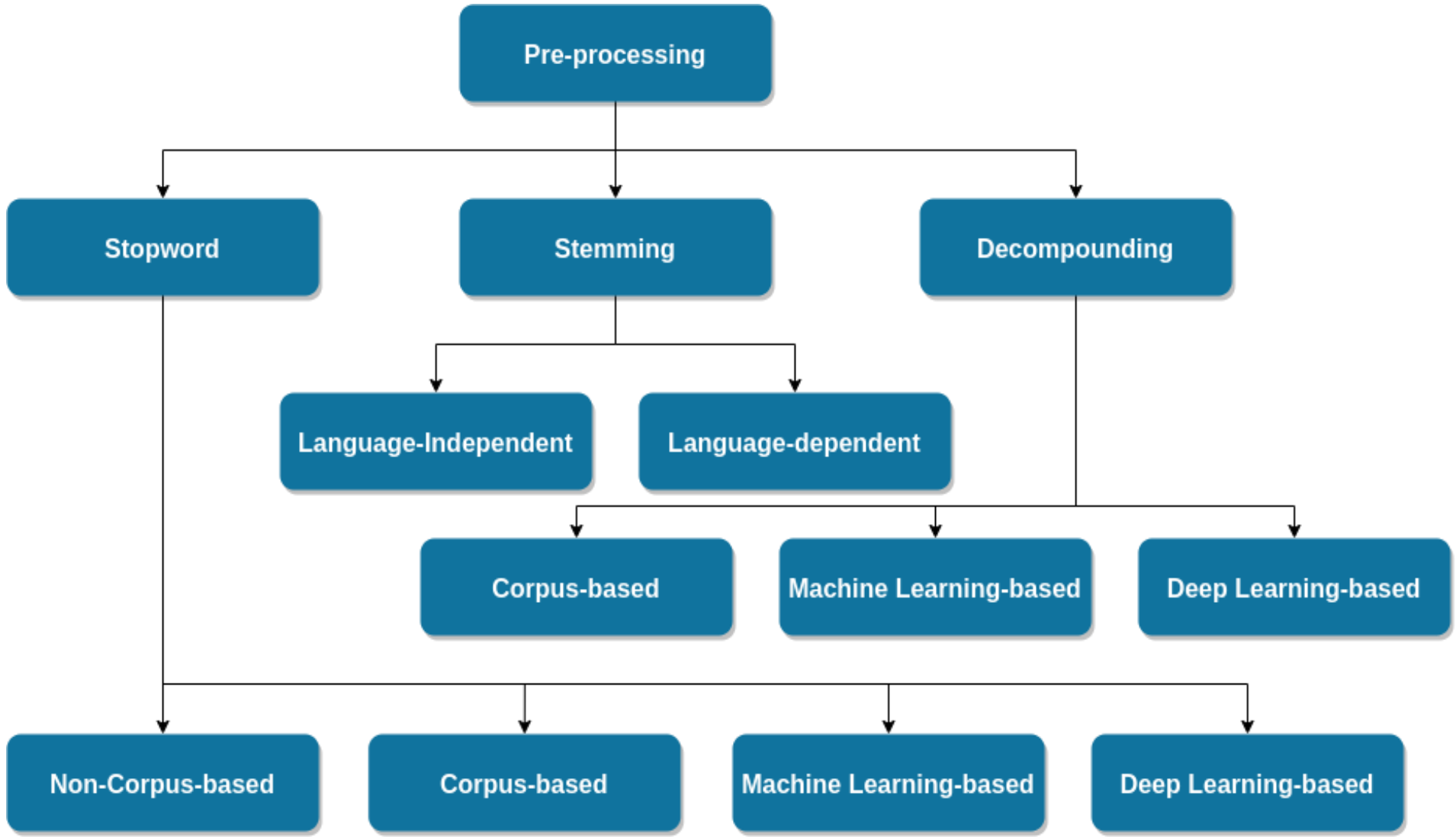
- Tokenization is unable to capture the meaning of the sentence hence, results in ambiguity.

- In certain languages like Chinese, Japanese, Arabic, lack distinct spaces between words. Hence, there is an absence of clear boundaries that complicates the process of tokenization.

- Text may also include more than one word, for example email address, URLs and special symbols, hence it is difficult to decide how to tokenize such elements.

Tokenisation issues

- 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. → ??


Pre-processing
Stopword
Stemming
Decompounding
Language-Independent
Language-dependent
Corpus-based
Machine Learning-based
Deep Learning-based
Non-Corpus-based
Corpus-based
Machine Learning-based
Deep Learning-based