Data Preprocessing:

- Data preprocessing is an essential step in building a Machine Learning model.
- Depending on how well the data has been preprocessed; the results are seen after training.

Text Preprocessing in Natural Language Processing

- Text preprocessing is the first step in the process of building a model.
- The text is represented as a vector in the multi-dimensional space.
- In the vector space model, each word/term is an axis/dimension. The number of unique words means the number of dimensions.
- Hence, Text preprocessing steps are widely used for dimensionality reduction.
- Various text preprocessing steps are:
    a. Tokenization
    b. Lower casing
    c. Stop words removal
    d. Stemming
    e. Lemmatization

Getting Started with NLTK

- NLTK stands for Natural Language Toolkit
- In this notebook, nltk library will be used to preprocess the text data
- Before beginning we need to install nltk, run below cell to install nltk

```
#install nltk
!pip install nltk
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: nltk in /home/anish/.local/lib/python3.8/site-packages (3.4.5)
Requirement already satisfied: six in /usr/lib/python3/dist-packages (from nltk) (1.14.0)
```

```
#browse the available packages
import nltk #import nltk
nltk.download()
```

```
showing info https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml

True
```

Select required module you want install, for example **book** in this case.
If you want to install all module, simply click on Download.

Now, lets load book module

```
#import all from book module
from nltk.book import *
```

Tokenization

- NLTK contains module called tokenize() which further classified into two categories
    - **Word Tokenization**
        - Word Tokenization simply means splitting sentences/text in words.
        - It is the process of splitting the given text into smaller pieces called tokens.
        - Words, numbers, punctuation marks, and others can be considered as tokens.
        - Use the **word_tokenize()** from **nltk.tokenize**
    - **Sentence Tokenization**
        - Sentence Tokenization is the process of splitting up strings into sentences.
        - A sentence usually ends with a full stop (.), here focus is to study the structure of sentences in the analysis.
        - use the **sent_tokenize()** from **nltk.tokenize**
        - Visit below for more:
            - [Tokenization_by_analyticsvidhya]
        - Let's start by practical example for **word tokenization**,

```
#word tokenization example using NLTK

#import word_tokenize from nltk
from nltk.tokenize import word_tokenize

#define sentence
sentence = """Founded in 2002, SpaceX's mission is to enable humans to become a spacefaring civilization and a multi
species by building a self-sustaining city on Mars. In 2008, SpaceX's Falcon 1 became the first privately developed
liquid-fuel launch vehicle to orbit the Earth."""

#tokenize words
words = word_tokenize(sentence)

print(words)
```

```
['Founded', 'in', '2002', ',', 'SpaceX', '’', 's', 'mission', 'is', 'to', 'enable', 'humans', 'to', 'become', 'a',
'spacefaring', 'civilization', 'and', 'a', 'multi-planet', 'species', 'by', 'building', 'a', 'self-sustaining', 'c
ity', 'on', 'Mars', '.', 'In', '2008', ',', 'SpaceX', '’', 's', 'Falcon', '1', 'became', 'the', 'first', 'privatel
y', 'developed', 'liquid-fuel', 'launch', 'vehicle', 'to', 'orbit', 'the', 'Earth', '.']
```

```
#sentence tokenization example using nltk

#import sent_tokenize from nltk
from nltk.tokenize import sent_tokenize

text = """Founded in 2002, SpaceX's mission is to enable humans to become a spacefaring civilization and a multi-pla
species by building a self-sustaining city on Mars. In 2008, SpaceX's Falcon 1 became the first privately developed
liquid-fuel launch vehicle to orbit the Earth."""

#tokenize sentence
sentences = sent_tokenize(text)

print(sentences)
```

```
['Founded in 2002, SpaceX's mission is to enable humans to become a spacefaring civilization and a multi-planet \n
species by building a self-sustaining city on Mars.', 'In 2008, SpaceX's Falcon 1 became the first privately devel
oped \nliquid-fuel launch vehicle to orbit the Earth.']
```

Lower Casing

- Converting word to lowercase (NLP->nlp).
- **Why Lower Casing?**
    - Words like **Book** and **book** mean the same,
    - When not converted to the lower case those two are represented as two different words in the vector space model (resulting in more dimension).
    - Higher the dimension, more computation resources are required.

```
#lower casing
sentence = "Books are on the table."
sentence = sentence.lower()
print(sentence)
```

```
books are on the table.
```

Stop words removal

- In natural language processing, useless words (data), are referred to as **stop words.**
- stop words are very commonly used words(a, an, the, etc.) in the documents.
- These kinds of words do not signify any importance as they do not help in distinguishing two documents.
- NLTK in python has a list of stopwords stored in 16 different languages.
- **Code Snippet:**

```
#stop words removal
from nltk.corpus import stopwords

sentence = "NLTK is a leading platform for building Python programs to work with human language data."

stop_words = set(stopwords.words('english'))
word_tokens = word_tokenize(sentence)

filtered_sentence = [word for word in word_tokens if word not in stop_words]

print("**Filtered words aafter removing the stop words from the given sentence is:**")
filtered_sentence
```

```
**Filtered words aafter removing the stop words from the given sentence is:**

['NLTK',
 'leading',
 'platform',
 'building',
 'Python',
 'programs',
 'work',
 'human',
 'language',
 'data',
 '.']
```

Stemming

- Stemming is a process of transforming a word to its root form.
- Stemming reduces the words _"chocolates", "chocolatey", "choco"_ to the root word, *"chocolate"*, and *"retrieval"*, *"retrieved"*, *"retrieves"* to the stem *"retrieve"*.
- Stemming is an important part of the pipelining process in Natural Language Processing.
- Input to the Stemming is tokenized words.
- Result obtained after stemming i.e. stem may or may not have meaning.
- Some more example of Stemming for:
    - root word _"like"_ include:
        - "Likes"
        - "Liked"
        - "Likely"
        - "Liking"
    - word _"histori"_ include:
        - History
        - Historical
    - word _"fina"_ include:
        - "Finally"
        - "final"
        - "Finalized"
    - word _"go"_ include:
        - "Going"
        - "Goes"
        - "gone"

Errors in Stemming

- **Over stemming**
    - It is the process where a much larger part of a word is chopped off than what is required, which in turn leads to two or more words being reduced to the same root word or stem incorrectly when they should have been reduced to two or more stem words.
    - Example:
        - _*University*_ and _*universe*_
        - Some stemming algorithms may reduce both the words to the stem univers, which would imply both the words mean the same thing, and that is clearly wrong.
        - Refers to false-positives
- **Under stemming**

- Under stemming, two or more words could be wrongly reduced to more than one root word, when they actually should be reduced to the same root word.
- Example:
  - consider the words "data" and "datum."
  - Some algorithms may reduce these words to dat and datu respectively, which is obviously wrong.
  - Both of these have to be reduced to the same stem dat.

```
#stemming words
from nltk.stem import PorterStemmer

ps = PorterStemmer()

sentence = "Machine Learning is cool"

#tokenize
words = sentence.split()

#stemming
for word in words:
    print(ps.stem(word))
```

```
machin
learn
is
cool
```

Explanation:

- The word 'machine' has its suffix 'e' chopped off.
- The stem does not make sense as it is not a word in English.
- This is a disadvantage of stemming.

Lemmatization

- Unlike stemming, lemmatization reduces the words to a word existing in the language.
- Libraries such as nltk, spacy have stemmers and lemmatizers implemented.
- Example of wordnet lemmatizer is as shown.
  - Wordnet is a large, freely and publicly available lexical database for the English language aiming to establish structured semantic(relating to meaning in language or logic) relationships between words.
  - It is most commonly used lemmatizers
- **Code Snippet:**

```
#import wordnet lemmatizer
from nltk.stem import WordNetLemmatizer

#Init the Wordnet Lemmatizer
lemmatizer = WordNetLemmatizer()

#lemmatize single word
print("Lemmatization of word **bats** is:", lemmatizer.lemmatize('bats'))
print("Lemmatization of word **are** is:", lemmatizer.lemmatize('are'))
print("Lemmatization of word **feet** is:", lemmatizer.lemmatize('feet'))
```

```
Lemmatization of word **bats** is: bat
Lemmatization of word **are** is: are
Lemmatization of word **feet** is: foot
```

```
#Lemmatize simple sentence
import nltk
"""
1. First tokenize the sentence into words, nltk.word_tokenize
2. call lemmatizer.lemmatize() on each word, can be one through list comprehension
"""

# Define the sentence to be lemmatized
sentence = "The striped bats are hanging on their feet for best"

# Tokenize: Split the sentence into words
word_list = nltk.word_tokenize(sentence)

# Lemmatize list of words and join
lemmatized_output = ' '.join([lemmatizer.lemmatize(w) for w in word_list])
print(lemmatized_output)
```

```
The striped bat are hanging on their foot for best
```

- Above code is simple example of how to use the wordnet lemmatizer on words and sentences.
- Notice it didn't do a good job. Because, 'are' is not converted to 'be' and 'hanging' is not converted to 'hang' as expected
- This can be corrected if we provide the correct (POS tag) as the second argument to **lemmatize()**.
- Sometimes, the same word can have a multiple lemmas based on the meaning/context

```
#using pos tag
print(lemmatizer.lemmatize("stripes", "v"))

print(lemmatizer.lemmatize("stripes", "n"))
```

```
strip
stripe
```

As seen above lemmas are different based on POS tagging

Problem:

- It may not be possible manaually provide the correct POS tag for every word for large texts.

solution:

- So, instead, we will find out the correct POS tag for each word, map it to the right input character that the WordnetLemmatizer accepts and pass it as the second argument to lemmatize()

Q. How to get POS tag for a given word?

- In nltk, it is available through the **nltk.pos_tag()** method
- **nltk.pos_tag()** only accepts a list of words i.e. list, even if its a single word.

```python
# Lemmatize with POS Tag
from nltk.corpus import wordnet

def get_wordnet_pos(word):
    """Map POS tag to first character lemmatize() accepts"""
    tag = nltk.pos_tag([word])[0][1][0].upper()
    tag_dict = {"J": wordnet.ADJ,
                "N": wordnet.NOUN,
                "V": wordnet.VERB,
                "R": wordnet.ADV}

    return tag_dict.get(tag, wordnet.NOUN)


# 1. Init Lemmatizer
lemmatizer = WordNetLemmatizer()

# 2. Lemmatize Single Word with the appropriate POS tag
word = 'feet'
print(lemmatizer.lemmatize(word, get_wordnet_pos(word)))

# 3. Lemmatize a Sentence with the appropriate POS tag
sentence = "The striped bats are hanging on their feet for best"
print([lemmatizer.lemmatize(w, get_wordnet_pos(w)) for w in nltk.word_tokenize(sentence)])
#> ['The', 'strip', 'bat', 'be', 'hang', 'on', 'their', 'foot', 'for', 'best']
```

```
foot
['The', 'strip', 'bat', 'be', 'hang', 'on', 'their', 'foot', 'for', 'best']
```

Hence, To improve the performance of lemmatization, we need to find the part of speech for each word in given string.

Difference between Stemming and Lemmatization

- Both Stemming and Lemmatization reduces the words to its base forms but,
    - Lemmatization considers the context and converts the word to its meaningful base form, whereas stemming just removes the last few characters, often leading to incorrect meaning and spelling errors.
    - In Lemmatization obtained base forms will make sense i.e. has meaning. Base form will be an actual word.
    - While incase of Stemming, the obtained base words may not have always meaning, it may be some non-existing word in the dictionary.
- Stemming is widely used, users can also make their own stemmer algorithm, whereas Lemmatization is not widely used, as it requires a lot of work.
    - stemming algorithm is found a lot in different languages
    - Lemmatization algorithm is not found much and is much more difficult and requires language knowledge as well.
- Stemmer is easier to build than a lemmatizer as the latter requires deep linguistics knowledge in constructing dictionaries to look up the lemma of the word.
- Example:
    - "Caring" -> Lemmatization -> "Care"
    - "Caring" -> Stemming -> "Car"
- Hence,

- In Stemming base form may or may not make sense
- In Lemmatization base form always make sense

Removal of Symbols and numbers

- Text data often contains symbols and numbers which are not quite useful for the purpose of analysis
- Regular Expression: sub()
- use to replace particular pattern of string in text by another string
- For more about regex visit:
  - Regex_w3_school
- **Code Snippet:**

```
#import regular expression
import re

#example-1
str = '875-873-2345 # commentss'

#remove comments
re_obj = re.sub(r'#.*$', '', str).strip()

#show
print("removed comments", re_obj)

#remove hyphens from re_obj
#\D returns match from string where the string doesnot contain numbers
re_hyphen_removed = re.sub(r'\D', '', re_obj).strip()
print("\nstring after removing hyphens:", re_hyphen_removed)
```

```
removed comments 875-873-2345

string after removing hyphens: 8758732345
```

```
#example-2
string ="""U.S. stock-index futures pointed
to a solidly higher open on Monday,
indicating that major
benchmarks were poised to USA reboundfrom last week's sharp decline,
\nwhich represented their biggest weekly drops in months."""

#replace "U.S.", "US", "USA" to "United States"
# | symbol indicates or
print(re.sub(r'U.S.|US|USA', 'United States', string))
```

```
United States stock-index futures pointed
to a solidly higher open on Monday,
indicating that major
benchmarks were poised to United StatesA reboundfrom last week's sharp decline,

which represented their biggest weekly drops in months.
```