Practical No. 1

Detail study of installation of various libraries in Python related to NLP like: NLTK, Gensim, CoreNLP, spaCy, TextBlob, Pattern, PyNLPl, etc.

Class: Final Year (ECO) Batch: _____

Roll No. _____ Date: _____

Objectives

- To understand the role of different Python libraries in implementing Natural Language Processing tasks.
- To learn the step-by-step installation process of popular NLP libraries such as NLTK, Gensim, CoreNLP, spaCy, TextBlob, Pattern, PyNLPl, etc.
- To explore the basic functionalities and features provided by each library.
- To verify successful installation and perform basic import tests in Python.
- To compare the scope, strengths, and limitations of different NLP libraries.

Theory

Natural Language Processing (NLP) refers to the field at the intersection of computer science, artificial intelligence, and linguistics. Its main goal is to enable computers to understand, interpret, and generate human language in a way that is valuable. Various Python libraries have been developed to aid in the processing and understanding of human language.

Installation Guide

i. **NLTK (Natural Language Toolkit)**

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

```bash
pip install nltk
```

ii. **Gensim**

Gensim is used for topic modeling and document similarity analysis.

```bash
pip install gensim
```

iii. **CoreNLP**

CoreNLP is a Java-based NLP toolkit developed by Stanford. To use it with Python, you'll also need the stanfordnlp or stanza library.

```bash
pip install stanfordnlp
```

Or

```bash
pip install stanza
```

iv.**spaCy**

spaCy is an open-source library for advanced natural language processing.

```bash
pip install spacy
```

For English language model:

```bash
python -m spacy download en_core_web_sm
```

v. **TextBlob**

TextBlob is a simple NLP library built on NLTK and Pattern.

```bash
python -m textblob.download_corpora
```

vi. **Pattern**

Pattern is a web mining module for the Python programming language.

```bash
pip install pattern
```

vii. **PyNLPl (or Pynlpl)**

PyNLPl is a Python library for natural language processing.

```bash
pip install pynlpl
```

Conclusion

Post Lab Questions

Solve any 5

1. List any three differences between **NLTK** and **spaCy** in terms of functionality and performance.
2. Mention two NLP tasks for which **Gensim** is commonly used.
3. What is the purpose of **Stanford CoreNLP**, and how is it accessed in Python?
4. Identify any two advantages of using **TextBlob** for text analysis.
5. Which NLP libraries provide support for processing Indian languages, and give one example of a task that can be performed using them?

Practical No. 2

Write a Python program to perform the pre-processing tasks like: tokenization (Word, Sentence), stemming, lemmatization, stop word removal etc. using primitive functions in NLTK.

Class: Final Year (ECO) Batch: _____

Roll No. _____ Date: _____

Objectives

- To understand the importance of text pre-processing in Natural Language Processing tasks.
- To learn and implement sentence tokenization and word tokenization using NLTK.
- To apply stemming and lemmatization for text normalization.
- To identify and remove stop words from text data.
- To gain hands-on experience with primitive NLTK functions for various pre-processing tasks.

Theory

Text pre-processing is a fundamental step in Natural Language Processing (NLP) that prepares raw text for analysis.
It involves cleaning and transforming text into a standard format suitable for computational processing.
Common pre-processing tasks include:

1. **Tokenization** – Splitting text into smaller units (tokens).
    - **Sentence Tokenization**: Breaking text into sentences.

- **Word Tokenization**: Breaking sentences into words.

2. **Stemming** – Reducing words to their root form by removing suffixes and prefixes, often without considering linguistic correctness. Example: “playing” → “play”.
3. **Lemmatization** – Reducing words to their base form (lemma) using vocabulary and morphological analysis. Example: “better” → “good”.
4. **Stop Word Removal** – Eliminating common words (like “is”, “the”, “and”) that carry little meaning in analysis.

The **Natural Language Toolkit (NLTK)** library in Python provides built-in functions to perform these tasks efficiently using primitive functions like word_tokenize(), sent_tokenize(), PorterStemmer, WordNetLemmatizer, and stopwords.

Procedure

1. Install the required NLTK library.
2. Import the necessary NLTK modules for tokenization, stemming, lemmatization, and stop word removal.
3. Download the required NLTK datasets such as punkt, wordnet, and stopwords.
4. Prepare a sample multi-sentence text for performing pre-processing tasks.
5. Perform **sentence tokenization** to split the text into sentences.
6. Perform **word tokenization** to split the sentences into individual words.
7. Apply **stemming** to convert each word to its root form using a stemming algorithm.
8. Apply **lemmatization** to get the base or dictionary form of each word.
9. Remove **stop words** from the tokenized list to keep only meaningful words.
10. Display the results for each stage of pre-processing.

Conclusion

_____

_____

_____

_____

Post Lab Questions

1. Differentiate between **stemming** and **lemmatization** with examples.
2. Explain the importance of **stop word removal** in NLP.
3. What is the role of the **punkt** dataset in NLTK tokenization?
4. How does lemmatization improve the quality of text pre-processing compared to stemming?
5. Give two real-life applications where tokenization is the first step in NLP processing.

Practical No. 3

Write a Python program to identify duplicate words in the given input text.

Class: Final Year (ECO) Batch: _____

Roll No. _____ Date: _____

Objectives

- To understand the process of reading and extracting text from external files in Python.
- To learn file handling operations for text-based data analysis.
- To perform text normalization for accurate comparison of words.
- To identify and list duplicate words present in a given input text file.

.

Theory

Duplicate word detection is a basic but important text preprocessing task in Natural Language Processing (NLP). It is used to identify and remove repeated words in a given text, which improves the quality of data before it is processed for further analysis. This task can be broken down into several key steps:

1. **File Handling in Python**
    - Text data is often stored in files. Python provides built-in methods like open(), read(), and readlines() to read content from files.
    - For example, open("file.txt", "r") allows reading the content line by line or all at once.
2. **Text Normalization**
    - Words in a text may appear in different cases (e.g., "Data" and "data").
    - To compare words accurately, all characters are converted to a common case,

usually lowercase.

3. **Tokenization**
    - Tokenization is the process of splitting the text into smaller units called tokens.
    - In this case, tokens are words, and tokenization helps separate the text into word-level units for analysis.
    - This can be done using Python's built-in split() method or using regular expressions (re.findall()) to extract only word characters and ignore punctuation.
4. **Duplicate Word Detection**
    - Once words are tokenized, a frequency count can be used to determine which words occur more than once.
    - Data structures such as **sets** and **dictionaries** are efficient for this task:
        - A set can store unique words encountered so far.
        - A dictionary can store each word as a key and its frequency as the value.
5. **Applications in NLP**
    - **Plagiarism Detection:** Identifying repeated words and phrases in documents.
    - **Data Cleaning:** Removing redundancies before machine learning training.
    - **Keyword Analysis:** Understanding repetitive terms in content to find emphasis or bias.

By combining file handling, normalization, tokenization, and duplicate detection, we can create a program that extracts text from a file and efficiently finds repeated words.

Stepwise Procedure

1. **Create a text file**
    - Save a file containing sample text with some repeated words.

2. **Open and read the file**
    - Use Python file handling (open() in read mode) to extract its contents.
3. **Normalize the text**
    - Convert the entire text to lowercase to avoid case-sensitive mismatches.
4. **Tokenize the text**
    - Split the text into words using regular expressions or the split() method.
5. **Identify duplicate words**
    - Use a set or dictionary to track word occurrences.
    - Words with a count greater than 1 are considered duplicates.
6. **Display the result**
    - Print the list of duplicate words and their frequencies.

Conclusion

_____

_____

_____

_____

Post Lab Questions

1. Explain the steps involved in reading text data from a file and how this input is further processed to identify duplicate words in Python.
2. Describe the role of regular expressions in extracting words from the given text.
3. How does converting the text to lowercase help in accurate duplicate detection?
4. Discuss the importance of using a dictionary or counter for storing and counting word frequencies.
5. Suggest modifications to the program to ignore common stop words while detecting duplicates.

Practical No. 4

Write a Python program to perform shallow parsing/chunking operation using customized grammar

Class: Final Year (ECO) Batch: _____

Roll No. _____ Date: _____

Objectives

- Understand the concept of shallow parsing (chunking) in Natural Language Processing.
- Implement chunking in Python using the nltk library.
- Define and apply a customised chunk grammar using regular expressions.
- Interpret chunked output for further NLP tasks.
- Differentiate between POS tagging and chunking.

Theory

Shallow Parsing (Chunking)

Shallow parsing, also known as chunking, is a Natural Language Processing technique used to identify and group related words in a sentence into higher-level units called *chunks*. Unlike full parsing, which produces a complete syntactic tree, shallow parsing focuses only on partial structures, such as **noun phrases (NP)**, **verb phrases (VP)**, or **prepositional phrases (PP)**.
Before chunking, each word in the text is assigned a Part-of-Speech (POS) tag indicating its grammatical category (noun, verb, adjective, etc.). Using these POS tags, chunking applies a **pattern-matching grammar** defined through regular expressions to extract meaningful phrases.

Customized Grammar

Chunk grammars are usually expressed using regular expressions over POS tags. These rules define which sequence of tags should be grouped together. For example, a noun phrase grammar may specify:

- An optional determiner (DT)
- Zero or more adjectives (JJ)
- A singular or plural noun (NN/NNS)

Such patterns allow the grouping of words into structured segments that can be used for downstream NLP tasks.

Key Features of Chunking

- Operates at a phrase level, not the full sentence structure.
- Uses POS tagging as an essential prerequisite.
- Allows customisation for domain-specific requirements.
- Can extract overlapping or nested phrases if grammar permits.

Applications

- Information extraction from text.
- Named Entity Recognition (NER).
- Building question-answering systems.
- Text summarisation.
- Preprocessing step for machine translation.

Stepwise Procedure

1. Install and import the NLTK library.
2. Define the input sentence.
3. Tokenize the sentence into words.
4. Perform POS tagging on the tokens.
5. Define a custom chunk grammar using regular expressions.
6. Create a RegexpParser object with the grammar.
7. Apply the parser to the POS-tagged sentence to obtain the chunk structure.
8. Display or visualize the chunks.

Conclusion

_____

_____

_____

_____

Post Lab Questions

1. Explain the difference between POS tagging and chunking.
2. Modify the grammar to detect verb phrases starting with a modal verb.
3. How does chunking differ from dependency parsing?
4. Mention an application where chunking is used in real-world NLP systems.