NLP Practical Write-up

Prac 1:- Perform tokenization (Whitespace, Punctuation-based, Treebank, Tweet, MWE) using NLTK library. Use porter stemmer and snowball stemmer for stemming. Use any technique for lemmatization. in python

Software Requirements:

1. Python (3.x recommended)
2. NLTK library (Natural Language Toolkit)
3. Required NLTK corpora and resources (punkt, stopwords, wordnet)

Hardware Requirements:

1. A computer with sufficient RAM and storage to run Python and NLTK comfortably.
2. Internet connectivity to download NLTK resources (if not already available).

Theory:

- **Tokenization:** Tokenization is the process of breaking down text into smaller units called tokens. NLTK provides various tokenization techniques, including whitespace, punctuation-based, Treebank, and tweet tokenization.
- **Stemming:** Stemming is the process of reducing words to their root or base form. It removes suffixes from words to obtain their stems. The NLTK library offers stemmers such as the Porter Stemmer and the Snowball Stemmer.
- **Lemmatization:** Lemmatization is similar to stemming but aims to obtain the base or dictionary form of a word, known as the lemma. NLTK provides the WordNetLemmatizer, which uses WordNet's lexical database to find the lemma of a word.

Procedure:

1. Import the necessary libraries: NLTK, string.
2. Download NLTK resources (if not already downloaded).
3. Define a sample text for tokenization.
4. Perform tokenization using various NLTK tokenizers: Whitespace Tokenizer, Punctuation-based Tokenizer, Treebank Tokenizer, Tweet Tokenizer.
5. Implement Multi-Word Expression (MWE) tokenization.
6. Perform stemming using Porter Stemmer and Snowball Stemmer.
7. Perform lemmatization using WordNetLemmatizer.

Conclusion: In this lab practical, we have explored various tokenization techniques provided by NLTK, including whitespace, punctuation-based, Treebank, and tweet tokenization. We have also implemented stemming using both Porter and Snowball stemmers, and lemmatization using the WordNetLemmatizer.

Prac 2. Perform bag-of-words approach (count occurrence, normalized count occurrence), TF-IDF on data. Create embeddings using Word2Vec. using python

Software Requirements:

1. Python (3.x recommended)
2. Necessary Python libraries: NLTK, Scikit-learn, Gensim
3. NLTK data (tokenizers, stopwords, WordNet)
4. Gensim's Word2Vec model

Hardware Requirements:

1. A computer with sufficient RAM and storage to run Python and the required libraries comfortably.
2. Internet connectivity to download NLTK data and Gensim's Word2Vec model (if not already available).

Theory:

- **Bag-of-Words (BoW):** BoW is a simple technique for text representation where each document is represented by a vector containing the frequency of each word occurring in the document. CountVectorizer from Scikit-learn is used to implement BoW.
- **TF-IDF:** TF-IDF is a statistical measure used to evaluate the importance of a word in a document relative to a collection of documents. It combines term frequency (TF) and inverse document frequency (IDF). TfidfVectorizer from Scikit-learn is used to implement TF-IDF.
- **Word Embeddings (Word2Vec):** Word embeddings are dense vector representations of words in a high-dimensional space. Word2Vec is a popular technique for generating word embeddings by training neural networks on large text corpora. Gensim library provides an implementation of Word2Vec.

Procedure:

1. Import the necessary libraries: NLTK, Scikit-learn, Gensim.
2. Download NLTK data (if not already downloaded).
3. Define sample text data.
4. Preprocess the text data by tokenizing, removing stopwords, punctuation, and lemmatizing.
5. Implement Bag-of-Words (BoW) using CountVectorizer from Scikit-learn.
6. Implement TF-IDF using TfidfVectorizer from Scikit-learn.
7. Generate word embeddings using Word2Vec from Gensim.

Conclusion: In this lab practical, we have explored different techniques for representing text data and creating word embeddings.

Prac 3 Perform text cleaning, perform lemmatization (any method), remove stop words (any method), label encoding. Create representations using TF-IDF. Save outputs.

Software Requirements:

1. Python (3.x recommended)
2. Necessary Python libraries: pandas, NLTK, Scikit-learn

3. NLTK data (for tokenization, stopwords, WordNet)
4. Storage space to save output files

Theory:

- **Text Cleaning:** Text cleaning involves preprocessing textual data by removing unnecessary elements such as punctuation, stop words, and converting text to lowercase.
- **Lemmatization:** Lemmatization is the process of reducing words to their base or dictionary form. It helps in standardizing words and reducing the vocabulary size.
- **Stop Words Removal:** Stop words are common words that do not carry significant meaning in a text. Removing stop words helps in focusing on the important words in the text.
- **Label Encoding:** Label encoding is used to convert categorical labels into numerical format, which is required for many machine learning algorithms.
- **TF-IDF Representation:** TF-IDF is a statistical measure used to evaluate the importance of a word in a document relative to a collection of documents. It combines term frequency (TF) and inverse document frequency (IDF).

Procedure:

1. Import the necessary libraries: pandas, NLTK, Scikit-learn.
2. Download NLTK data (if not already downloaded).
3. Define sample textual data.
4. Perform text cleaning by converting text to lowercase, tokenizing, removing stop words, punctuation, and lemmatizing.
5. Encode the labels into numerical format using LabelEncoder from Scikit-learn.
6. Create TF-IDF representations of the preprocessed text data using TfidfVectorizer from Scikit-learn.
7. Save the preprocessed data and TF-IDF representations to files.

Conclusion: In this lab practical, we have successfully preprocessed textual data. encode the labels also we have created representations of the text data using the TF-IDF technique,

Prac 4. Create a transformer from scratch using the Pytorch librar

Software Requirements:

1. Python (3.x recommended)
2. PyTorch library
3. Any text editor or IDE (e.g., VSCode, Jupyter Notebook) for coding

Theory:

- **Transformer Architecture:** The Transformer model consists of an encoder-decoder architecture, where both the encoder and decoder are composed of multiple layers of self-attention mechanisms and feed-forward neural networks.
- **Self-Attention Mechanism:** Self-attention allows the model to weigh the importance of different words in a sequence when encoding or decoding. It computes attention

scores between each pair of words in a sequence and aggregates them to obtain context-aware representations.

- **Positional Encoding:** Since Transformer models do not inherently have positional information like recurrent neural networks, positional encodings are added to the input embeddings to provide information about the position of each word in the sequence.
- **Encoder:** The encoder of the Transformer model processes the input sequence and produces a sequence of hidden representations. Each layer of the encoder contains a self-attention mechanism followed by a feed-forward neural network.
- **Decoder:** The decoder takes the encoder's output and generates the output sequence. Similar to the encoder, each layer of the decoder consists of a self-attention mechanism and a feed-forward neural network. Additionally, the decoder utilizes an encoder-decoder attention mechanism to attend to the input sequence.

Procedure:

1. Import the necessary libraries: PyTorch.
2. Implement the SelfAttention module, which computes the self-attention mechanism.
3. Implement the TransformerBlock module, which represents a single layer of the Transformer model.
4. Implement the Encoder module, which consists of multiple TransformerBlock layers to encode the input sequence.
5. Implement the Transformer module, which combines the encoder and decoder components of the Transformer architecture.
6. Define any additional helper functions for data preprocessing, training, and evaluation if needed.
7. Train the Transformer model on a suitable dataset for a specific task, such as machine translation or text generation.

Conclusion: In this lab practical, we have successfully implemented a Transformer model from scratch using the PyTorch library.

Prac 5. **Morphology is the study of the way words are built up from smaller meaning bearing units. Study and understand the concepts of morphology by the use of add delete table using python**

Theory:

- **Morphology:** Morphology is a branch of linguistics that studies the internal structure of words and how they are formed from smaller units called morphemes. Morphemes are the smallest units of meaning in a language and can be added, removed, or changed to create new words or alter the meaning of existing words.
- **Add-Delete Table:** An add-delete table is a visual representation that illustrates how words change when specific morphemes are added or deleted. It helps in understanding the morphological processes and how they affect the meaning or grammatical function of words.

Procedure:

1. Define a sample set of words along with their variations that demonstrate changes in meaning or grammatical function when morphemes are added or deleted.
2. Create a Python script to generate the add-delete table.
3. Define a function to iterate through each word and its variations, add or delete morphemes, and record the resulting word.
4. Print the add-delete table for each word, its variations, and the operations performed (adding or deleting morphemes) along with the resulting word.
5. Analyze the add-delete table to understand how morphemes influence the structure and meaning of words.

Conclusion: In this lab practical, we have successfully studied the concepts of morphology by creating an add-delete table