TRIBHUVAN UNIVERSITY

Institute of Science and Technology (IoST)

Samriddhi College

A Lab Report on Artificial Intelligence

LAB 9: Natural Language Processing

Submitted To:

Bikram Acharya

Submitted By:

Kanchan Joshi (Roll no.10)

Date: October 04, 2025

1. Introduction

Natural Language Processing (NLP) is a subfield of artificial intelligence that focuses on the interaction between computers and human language. This laboratory explores fundamental NLP techniques, including text preprocessing, tokenization, stemming, lemmatization, and part-of-speech (POS) tagging, which are essential for tasks like sentiment analysis, machine translation, and chatbots.

The source code implementations and complete materials for the NLP systems examined in this laboratory work are maintained in an accessible repository for future reference.

NLP bridges the gap between human communication and machine understanding by processing unstructured text data. Key challenges include ambiguity, context dependency, and syntactic complexity, addressed through statistical and deep learning models.

1.1 NLP Pipeline Overview

A typical NLP pipeline consists of:

1. **Tokenization**: Splitting text into words or sentences.
2. **Stemming/Lemmatization**: Reducing words to their base form.
3. **POS Tagging**: Assigning grammatical tags to words.
4. **Named Entity Recognition (NER)**: Identifying entities like persons or locations.

These steps enable higher-level applications such as text classification and question answering.

1.2 Challenges in NLP

NLP faces several significant challenges that make it a complex and evolving field. One major issue is ambiguity, where words or phrases can have multiple

meanings depending on the context-for instance, the word "bank" could refer to a financial institution or the side of a river, and computers need to discern which one fits based on surrounding words. Another hurdle is context dependency; sentences like "I saw her duck" could mean a bird or an action, requiring an understanding of prior or following text. Sarcasm and irony add layers of difficulty, as they often convey the opposite of literal meaning, which is tough for models to detect without cultural or emotional cues. Multilingual support is also challenging, as languages vary in structure, script, and idioms, making a one-size-fits-all approach impractical. Finally, computational efficiency is a concern, especially with massive datasets, demanding optimized algorithms to process text in real-time.

Modern transformer models, such as BERT or GPT, address these by using self-attention mechanisms that allow the model to weigh the importance of different words in a sentence relative to each other, capturing long-range dependencies that traditional recurrent neural networks (RNNs) struggle with. They also incorporate bidirectional context, meaning they look at both what comes before and after a word, leading to better disambiguation and contextual understanding, ultimately boosting accuracy in tasks like sentiment detection or translation.

2. NLP Implementation

This implementation uses Python's NLTK library for core NLP tasks, demonstrating text preprocessing on sample sentences.

2.1 Text Preprocessing Framework

```python
import nltk
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.stem import PorterStemmer, WordNetLemmatizer
from nltk import pos_tag
from nltk.corpus import stopwords

# Download required NLTK data
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')
nltk.download('stopwords')

def preprocess_text(text):
    """Preprocess text: tokenize, remove stopwords, stem/lemmatize"""
    # Sentence tokenization
    sentences = sent_tokenize(text)

    processed = []
    stemmer = PorterStemmer()
    lemmatizer = WordNetLemmatizer()
    stop_words = set(stopwords.words('english'))

    for sentence in sentences:
        # Word tokenization
        words = word_tokenize(sentence.lower())
        # Remove stopwords
```

```
        words = [w for w in words if w.isalpha() and w not in
            stop_words]
        # Stemming
        stemmed = [stemmer.stem(w) for w in words]
        # Lemmatization
        lemmatized = [lemmatizer.lemmatize(w) for w in words]
        # POS Tagging
        pos_tags = pos_tag(words)

        processed.append({
            'original': sentence,
            'tokens': words,
            'stemmed': stemmed,
            'lemmatized': lemmatized,
            'pos_tags': pos_tags
        })

    return processed
```

Code 2.1: Text Tokenization and Preprocessing

2.2 Demonstration Results

Sample text: "Natural language processing is a fascinating field in artificial intelligence."

Processed Output:

- Tokens: ['natural', 'language', 'processing', 'fascinating', 'field', 'artificial', 'intelligence']

- Stemmed: ['natur', 'languag', 'process', 'fascin', 'field', 'artifici', 'intellig']

- Lemmatized: ['natural', 'language', 'processing', 'fascinating', 'field', 'artificial', 'intelligence']

- POS Tags: [('natural', 'JJ'), ('language', 'NN'), ('processing', 'NN'), ...]

This demonstrates how preprocessing reduces text to meaningful features for analysis.

3. NLP Code Examples

3.1 Example 1: Text Cleaning and Tokenization

This example demonstrates cleaning text by removing URLs, punctuation, and converting to lowercase, followed by word tokenization.

```
import re
from nltk.tokenize import word_tokenize

# Example text
text = "Hello, world! Welcome to NLP. Visit https://example.com for
   more details."

# Remove URLs
cleaned_text = re.sub(r"http\S+", "", text)

# Remove punctuation and convert to lowercase
cleaned_text = re.sub(r"[^\w\s]", "", cleaned_text).lower()

print("Cleaned Text:", cleaned_text)

# Tokenizing words
tokens = word_tokenize(cleaned_text)
print("Tokens:", tokens)
```

Code 3.1: Text Cleaning and Tokenization

```
Cleaned Text: hello world welcome to nlp visit examplecom for more
   details

Tokens: ['hello', 'world', 'welcome', 'to', 'nlp', 'visit', '
   examplecom', 'for', 'more', 'details']
```

Code 3.2: Executed Output

3.2 Example 2: Removing Stop Words, Stemming, and Lemmatization

After tokenization, remove common stop words and reduce words to base forms using stemming and lemmatization.

```
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, WordNetLemmatizer

# Assuming tokens from previous example
tokens = ['hello', 'world', 'welcome', 'to', 'nlp', 'visit', '
    examplecom', 'for', 'more', 'details']

# Define English stop words
stop_words = set(stopwords.words('english'))

# Remove stop words
filtered_tokens = [word for word in tokens if word not in stop_words
    ]
print("Filtered Tokens:", filtered_tokens)

# Initialize stemmer and lemmatizer
stemmer = PorterStemmer()
lemmatizer = WordNetLemmatizer()

# Stemming
stemmed_words = [stemmer.stem(word) for word in filtered_tokens]

# Lemmatization
lemmatized_words = [lemmatizer.lemmatize(word) for word in
    filtered_tokens]

print("Stemmed Words:", stemmed_words)
print("Lemmatized Words:", lemmatized_words)
```

Code 3.3: Stop Words Removal, Stemming, and Lemmatization

```
Filtered Tokens: ['hello', 'world', 'welcome', 'nlp', 'visit', '
    examplecom', 'details']

Stemmed Words: ['hello', 'world', 'welcom', 'nlp', 'visit', '
    examplecom', 'detail']
```

```
4
5 Lemmatized Words: ['hello', 'world', 'welcome', 'nlp', 'visit', '
     examplecom', 'detail']
```

Code 3.4: Executed Output

3.3 Example 3: Bag of Words (BoW) Feature Extraction

Convert preprocessed text into numerical features using Bag of Words model.

```
from sklearn.feature_extraction.text import CountVectorizer

# Example sentences
documents = [
   "I love NLP and machine learning.",
   "NLP is the future of AI.",
   "Machine learning is a part of AI."
]

# Initialize the vectorizer
vectorizer = CountVectorizer()

# Fit and transform the documents
X = vectorizer.fit_transform(documents)

# Convert the result to an array
print("BoW Matrix:")
print(X.toarray())

# Display the feature names (words)
print("Feature Names:", vectorizer.get_feature_names_out())
```

Code 3.5: Bag of Words

```
BoW Matrix:
[[0 1 0 0 1 1 1 1 0 0 0]
 [1 0 1 1 0 0 0 1 1 0 1]
 [1 0 0 1 1 0 1 0 1 1 0]]
Feature Names: ['ai' 'and' 'future' 'is' 'learning' 'love' 'machine'
    'nlp' 'of' 'part'
 'the']
```

Code 3.6: Executed Output

3.4 Example 4: TF-IDF Feature Extraction

TF-IDF weights terms based on frequency and rarity across documents.

```python
from sklearn.feature_extraction.text import TfidfVectorizer

# Same documents as above
documents = [
    "I love NLP and machine learning.",
    "NLP is the future of AI.",
    "Machine learning is a part of AI."
]

# Initialize the TF-IDF vectorizer
tfidf_vectorizer = TfidfVectorizer()

# Fit and transform the documents
X_tfidf = tfidf_vectorizer.fit_transform(documents)

# Convert the result to an array
print("TF-IDF Matrix:")
print(X_tfidf.toarray())

# Display the feature names (words)
print("Feature Names:", tfidf_vectorizer.get_feature_names_out())
```

Code 3.7: TF-IDF

```
TF-IDF Matrix:
[[0. 0.51741994 0. 0. 0.3935112 0.51741994
  0.3935112 0.3935112 0. 0. 0. ]
 [0.36617957 0. 0.48148213 0.36617957 0. 0.
  0. 0.36617957 0.36617957 0. 0.48148213]
 [0.38550292 0. 0. 0.38550292 0.38550292 0.
  0.38550292 0. 0.38550292 0.50689001 0. ]]
Feature Names: ['ai' 'and' 'future' 'is' 'learning' 'love' 'machine'
    'nlp' 'of' 'part'
 'the']
```

Code 3.8: Executed Output

4. Discussion

NLP techniques enable machines to process human language, powering applications like virtual assistants and recommendation systems. The lab highlights preprocessing's role in feature extraction, but real-world NLP requires handling noise, dialects, and ethics (bias in models).

NLTK provides accessible tools for education, while production uses spaCy or Hugging Face Transformers for efficiency. Limitations include computational cost for large corpora; future directions involve multimodal NLP (text+image).

5. Conclusion

This laboratory demonstrates core NLP components, from tokenization to POS tagging, essential for AI language understanding. Implementations reveal the pipeline's modularity, while exam questions underscore theoretical foundations.

Future work could extend to advanced tasks like NER or sequence labeling using deep learning.