Dr.D.Y.Patil Institute of Technology,Pimpri,Pune-18

Department of Computer Engineering

# LP VI –Natural Language Processing Lab Manual

BE Sem II – 2024-25

| S.No | Assignment Title | Page No |
|---|---|---|
| 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. Input / Dataset –use any sample sentence | 2 |
| 2 | .Perform bag-of-words approach (count occurrence, normalized count occurrence), tf-idf on data. Create embeddings using Word2Vec. Dataset to be used: https://www.kaggle.com/datasets/CooperUnion/cardataset | 5 |
| 3 | Perform text cleaning, perform lemmatization (any method), remove stop words (any method), label encoding. Create representations using TF-IDF. Save outputs. Dataset: https://github.com/PICT-NLP/BE-NLP-Elective/blob/main/3-Preprocessing/News_dataset.pickle | 8 |
| 4 | Create a transformer from scratch using the Pytorch library | 12 |
| 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 | 15 |
| 6 | Mini Project (Fine tune transformers on your preferred task) Finetune a pretrained transformer for any of the following tasks on any relevant dataset of your choice: • Neural Machine Translation • Classification • Summarization Provide some basic theory and introduction on each of these above mentioned objectives, from the point of view of comceptual understanding(do not exceed more than 250 words for each) | 17 |

# Assignment no 01

**Problem Statement:-**

Perform tokenization (Whitespace, Punctuation-based, Treebank, Tweet, MWE) using nltk library. Use porter stemmer and snowball stemmer for stemming. Use any technique for lemmatization. Input / Dataset –use any sample sentence

**Solution:**

Tokenization is the process of breaking down text into smaller units called tokens, which can be words, phrases, or punctuation marks. Types include Whitespace Tokenization (splitting text by spaces), Punctuation-based Tokenization (splitting by punctuation), Treebank Tokenization (splitting by linguistic rules), Tweet Tokenization (handling social media text), and MWE Tokenization (splitting multi-word expressions).

Stemming reduces words to their base or root form using algorithms like Porter Stemmer and Snowball Stemmer, e.g., "running" becomes "run."

Lemmatization involves reducing words to their base form (lemma) considering the word's meaning and part of speech, e.g., "better" becomes "good."

### Task 1: Tokenization, Stemming, Lemmatization

```python
import nltk

from nltk.tokenize import word_tokenize, TreebankWordTokenizer, TweetTokenizer

from nltk.tokenize import MWETokenizer, WhitespaceTokenizer, WordPunctTokenizer

from nltk.stem import PorterStemmer, SnowballStemmer
```

```
from nltk.corpus import wordnet

from nltk.stem import WordNetLemmatizer

# Sample sentence

sentence = "The quick brown fox jumps over the lazy dog."

# Tokenization

whitespace_tokens = WhitespaceTokenizer().tokenize(sentence)

punctuation_tokens = WordPunctTokenizer().tokenize(sentence)

treebank_tokens = TreebankWordTokenizer().tokenize(sentence)

tweet_tokens = TweetTokenizer().tokenize(sentence)

mwe_tokens = MWETokenizer().tokenize(sentence.split())

# Stemming

porter_stemmer = PorterStemmer()

snowball_stemmer = SnowballStemmer('english')

porter_stems = [porter_stemmer.stem(token) for token in treebank_tokens]

snowball_stems = [snowball_stemmer.stem(token) for token in treebank_tokens]



# Lemmatization

lemmatizer = WordNetLemmatizer()
```

```
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)
lemmatized_tokens = [lemmatizer.lemmatize(token, get_wordnet_pos(token)) for token in treebank_tokens]
# Output
print("Whitespace Tokenization:", whitespace_tokens)
print("Punctuation-based Tokenization:", punctuation_tokens)
print("Treebank Tokenization:", treebank_tokens)
print("Tweet Tokenization:", tweet_tokens)
print("MWE Tokenization:", mwe_tokens)
print("Porter Stemming:", porter_stems)
print("Snowball Stemming:", snowball_stems)
print("Lemmatization:", lemmatized_tokens)
```

# Assignment No 02

**Problem Statement:-**

.Perform bag-of-words approach (count occurrence, normalized count occurrence), tf-idf on data. Create embeddings using Word2Vec. Dataset to be used: https://www.kaggle.com/datasets/CooperUnion/cardataset

**Solution:-**

The Bag-of-Words (BoW) approach represents text by the frequency of each word in the document. It captures the occurrence but ignores the order. Count occurrence counts word frequency, while normalized count occurrence scales the frequency.

TF-IDF (Term Frequency-Inverse Document Frequency) measures word importance in a document relative to the entire corpus. High TF-IDF indicates a word is significant in a specific document.

Word2Vec creates word embeddings by training on large corpora, capturing semantic relationships. It represents words as vectors in a continuous space where semantically similar words are closer together.

Code-

```python

import pandas as pd

from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer

from gensim.models import Word2Vec
```

```
# Load dataset

df = pd.read_csv('path_to_dataset.csv')  # Adjust path accordingly

# Sample text column

texts = df['text_column_name']  # Adjust column name accordingly

# Bag-of-Words

count_vectorizer = CountVectorizer()

count_occurrence = count_vectorizer.fit_transform(texts)

normalized_count_occurrence = count_occurrence.toarray() / count_occurrence.toarray().sum(axis=1, keepdims=True)

# TF-IDF

tfidf_vectorizer = TfidfVectorizer()

tfidf_matrix = tfidf_vectorizer.fit_transform(texts)

# Word2Vec

tokenized_texts = [word_tokenize(text) for text in texts]

word2vec_model = Word2Vec(sentences=tokenized_texts, vector_size=100, window=5, min_count=1, workers=4)
```

**Output**

```
print("Bag-of-Words Count Occurrence:", count_occurrence)

print("Normalized Count Occurrence:", normalized_count_occurrence)

print("TF-IDF Matrix:", tfidf_matrix)

print("Word2Vec Model:", word2vec_model.wv)
```

# Assignment No 03

**Problem Statement:-**

Perform text cleaning, perform lemmatization (any method), remove stop words (any method), label encoding. Create representations using TF-IDF. Save outputs. Dataset: https://github.com/PICT-NLP/BE-NLP-Elective/blob/main/3-Preprocessing/News_dataset.pickle

**Solution:-**

Introduction to terms

Text Cleaning involves preprocessing text, removing noise, and preparing it for analysis. This includes lemmatization (reducing words to their lemma), removing stop words (common words that add little value), and label encoding (converting categorical labels into numerical form).

TF-IDF representation transforms text into numerical features, emphasizing important words.

Code-

Let's use the provided dataset from GitHub.

```python
import pandas as pd

import nltk

from nltk.corpus import stopwords

from sklearn.preprocessing import LabelEncoder
```

```
from sklearn.feature_extraction.text import TfidfVectorizer

import pickle

# Load dataset

with open('path_to_news_dataset.pickle', 'rb') as file:

    news_dataset = pickle.load(file)

texts = news_dataset['text']

labels = news_dataset['label']

# Text Cleaning, Lemmatization

lemmatizer = WordNetLemmatizer()

stop_words = set(stopwords.words('english'))

def preprocess_text(text):

    tokens = word_tokenize(text.lower())

    tokens = [token for token in tokens if token.isalpha()]

    tokens = [token for token in tokens if token not in stop_words]

    tokens = [lemmatizer.lemmatize(token) for token in tokens]

    return ' '.join(tokens)

cleaned_texts = texts.apply(preprocess_text)



# Label Encoding
```

```python
label_encoder = LabelEncoder()

encoded_labels = label_encoder.fit_transform(labels)

# TF-IDF

tfidf_vectorizer = TfidfVectorizer()

tfidf_matrix = tfidf_vectorizer.fit_transform(cleaned_texts)

# Save Outputs

cleaned_texts.to_csv('cleaned_texts.csv', index=False)

pd.DataFrame(encoded_labels, columns=['label']).to_csv('encoded_labels.csv', index=False)

pd.DataFrame(tfidf_matrix.toarray(),
columns=tfidf_vectorizer.get_feature_names_out()).to_csv('tfidf_matrix.csv', index=False)
```

**Output**

print("Cleaned Texts:", cleaned_texts)

print("Encoded Labels:", encoded_labels)

print("TF-IDF Matrix:", tfidf_matrix)

```
```

# Assignment No 04

**Problem Statement:-**

Create a transformer from scratch using the Pytorch library

**Solution:-**

Task 4: Creating a Transformer from Scratch using PyTorch

Transformers are deep learning models for NLP tasks. They use self-attention mechanisms to capture contextual relationships in text. PyTorch provides the tools to build and train custom transformer models from scratch, enabling efficient parallel processing of text data.

Here's a simplified example of creating a transformer model using PyTorch.

```python
import torch

import torch.nn as nn

import torch.nn.functional as F

import torch.optim as optim

class TransformerModel(nn.Module):

    def __init__(self, input_dim, n_heads, num_layers, hidden_dim, output_dim):

        super(TransformerModel, self).__init__()

        self.embedding = nn.Embedding(input_dim, hidden_dim)
```

```
        self.transformer = nn.Transformer(hidden_dim, n_heads, num_layers)
        self.fc_out = nn.Linear(hidden_dim, output_dim)
    def forward(self, src):
        embedded = self.embedding(src)
        transformer_out = self.transformer(embedded)
        output = self.fc_out(transformer_out)
        return output
# Example usage
input_dim = 10000
n_heads = 8
num_layers = 3
hidden_dim = 512
output_dim = 1
model = TransformerModel(input_dim, n_heads, num_layers, hidden_dim, output_dim)
optimizer = optim.Adam(model.parameters(), lr=0.001)
criterion = nn.MSELoss()
# Example data (dummy)
src = torch.randint(0, input_dim, (10, 32))  # Sequence length x Batch size
target = torch.rand((10, 32, output_dim))
```

```
# Training loop
model.train()
for epoch in range(10):
    optimizer.zero_grad()
    output = model(src)
    loss = criterion(output, target)
    loss.backward()
    optimizer.step()
    print(f'Epoch {epoch+1}, Loss: {loss.item()}')
```

# Assignment No 05

**Problem Statement:-**

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

**Solution:-**

Task 5: Understanding Morphology

Morphology

Morphology is the study of word structure, focusing on how words are formed from morphemes (smallest meaning-bearing units). It examines root words, prefixes, suffixes, and inflections, helping in understanding word formation and grammar. An add/delete table is used to illustrate morphological changes by adding or removing morphemes.

Here's a brief explanation of morphology with an example of add/delete table:

Morphology is the study of how words are formed and their relationship to other words in the same language. It involves analyzing the structure of words, including roots, prefixes, suffixes, and inflections.

Add/Delete Table Example:

| Base Word | Add | Delete |
|-----------|-----|--------|
| Run | -s | |

| | | |
|---|---|---|
| Playing | -ing | -ing |
| Unhappiness | un-, -ness | un-, -ness |

# Assignment No 06

**Problem Statement:-**

Mini Project (Fine tune transformers on your preferred task) Finetune a pretrained transformer for any of the following tasks on any relevant dataset of your choice: • Neural Machine Translation • Classification • Summarization

Solution:-

Introduction:

Fine-tuning Transformers

Fine-tuning pre-trained transformer models, like BERT or GPT, involves adapting them to specific tasks such as Neural Machine Translation, Classification, or Summarization. This process enhances model performance by leveraging pre-learned knowledge and applying it to specific datasets and tasks, enabling more accurate and efficient results.

Code:-

You can choose any pre-trained transformer model (like BERT, GPT-3, T5) and fine-tune it for a specific task. Here's an example using the Hugging Face Transformers library for text classification:

```python

from transformers import BertTokenizer, BertForSequenceClassification, Trainer, TrainingArguments

import torch

from datasets import load_dataset
```

```
# Load dataset
dataset = load_dataset("imdb")

# Tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
tokenized_datasets = dataset.map(lambda x: tokenizer(x['text'], truncation=True, padding='max_length'), batched=True)
tokenized_datasets.set_format(type='torch', columns=['input_ids', 'attention_mask', 'label'])
# Model
model = BertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=2)
# Training
training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=3,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    warmup_steps=500,
    weight_decay=0.01,
```

```
    logging_dir='./logs',
)
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_datasets['train'],
    eval_dataset=tokenized_datasets['test'],
)
trainer.train()
```