FACULTY OF ENGINEERING AND TECHNOLOGY
BACHELOR OF TECHNOLOGY

DEEP LEARNING WITH N L P LABORATORY
(203105477)

7th SEMESTER

COMPUTER SCIENCE & ENGINEERING DEPARTMENT

LABORATORY MANUAL

CERTIFICATE

This is to certify that **Mr. VOLADRI SAIKIRAN** with enrolment no. **200303124537** has successfully completed his laboratory experiments in the **DEEP LEARNING WITH NLP LABORATORY** (203105477) from the department of CSE-AI during the academic year 2023-24

Date of Submission: ....

Staff In charge: ....

Head Of Department: ....

TABLE OF CONTENT

PRACTICAL – 1

AIM: - Implementation of preprocessing of Text with NLTK (Tokenization, Stemming, Lemmatization and removal of stop words in NLP.

TOKENIZATION: -

Tokenization refers to break down the text into smaller units. It entails splitting paragraphs into sentences and sentences into words. It is one of the initial steps of any NLP preprocessing.

CODE: -

```
text = "this is deep learning practical 1

"tokens = text.split()

print(tokens)
```

OUTPUT: -

```
['This', 'is', 'Deep', 'learning', 'Practical', '1']
PS C:\Users\saiki\OneDrive\Desktop\7th sem\LABS>
```

STEMMING: -

Stemming generates the base word from the word by removing the affixes of the word. It must be noted that stemmers might not always result in semantically meaningful base words.

CODE: -

```
import nltk
from nltk.stem import PorterStemmer
ps = PorterStemmer()
sentence = "this is deep learning practical 1 "
for word in sentence.split():
  print(ps.stem(word))
```

OUTPUT: -

```
thi
is
deep
learn
practic
1
PS C:\Users\saiki\OneDrive\Desktop\7th sem\LABS>
```

LEMMATIZATION: -

Lemmatization involves grouping together the inflected forms of the same word. This way, we can reach out to the base form of any word which will be meaningful in nature. The base from here is called the Lemma.

CODE: -

```
import nltk
nltk.download('wordnet')
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
print(lemmatizer.lemmatize("reading", pos= "v"))
print(lemmatizer.lemmatize("teaching", pos= "v"))
```

OUTPUT: -

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\saiki\AppData\Roaming\nltk_data...
read
teach
PS C:\Users\saiki\OneDrive\Desktop\7th sem\LABS>  c:; cd 'c:
```

REMOVAL OF STOP WORDS: -

Stop word removal is one of the most commonly used preprocessing steps across different NLP applications. The idea is simply removing the words that occur commonly across all the documents in the corpus.

CODE: -

```
import nltk

from nltk.corpus import stopwords

nltk.download('stopwords')

print(stopwords.words('english'))
```

OUTPUT: -

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\saiki\AppData\Roaming\nltk_data...
[nltk_data]   Unzipping corpora\stopwords.zip.
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himse
she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll
ese', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or',
use', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up
wn', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more',
, 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now',
'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'is
sn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'would
ouldn't"]
PS C:\Users\saiki\OneDrive\Desktop\7th sem\LABS> 
```

PRACTICAL - 2

AIM: - Implementation to Convert the text to word count vectors with ScikitLearn (CountVectorizer).

CODE: -

```
from sklearn.feature_extraction.text import CountVectorizer

corpus = ["Sometimes life can get confusing and hard",
          "If you don't sacrifice for what you want then what you want is your

sacrifice."]

vectorizer = CountVectorizer()

vectorizer.fit(corpus)

print("Vocabulary:" , vectorizer.vocabulary_)

vector = vectorizer.transform(corpus)

print("Encoded corpus is:")

print(vector.toarray()
```

OUTPUT:

```
Vocabulary: {'what': 12, 'you': 13, 'do': 1, 'today': 9, 'is': 5, 'that': 7, 'achieve': 0, 'tommorow': 10, 'if': 4, 'dont': 2, 'sacrifice': 6, 'for': 3, 'want': 11, 't
 14}
Encoded corpus is:
[[1 1 0 0 0 1 0 1 0 1 1 0 1 2 0]
 [0 0 1 1 1 1 2 0 1 0 0 2 2 3 1]]
PS C:\Users\saiki\OneDrive\Desktop\7th sem\LABS>
```

PRACTICAL – 3

AIM: - Implementation to Convert the text to word frequency vectors with ScikitLearn (TfidfVectorizer).

TFIDF: - TF-IDF is an abbreviation for Term Frequency Inverse Document Frequency. This is very common algorithm to transform text into a meaningful representation of numbers which is used to fit machine algorithm for prediction.

FORMULAS: -

$$\text{Frequency} = \frac{Number\ of\ terms\ repeat\ in\ the\ document}{Total\ number\ of\ terms\ in\ the\ document}$$

$$\text{IDF} = \frac{Number\ of\ documents\ in\ the\ corpus}{Total\ number\ of\ documents\ in\ the\ corpus\ contain\ term}$$

CODE: -

```
from sklearn.feature_extraction.text import TfidfVectorizer
corpus = ["Sometimes life can get confusing and hard",
          "In such times it can be useful to turn to the  wisdom of poetry."]

vectorizer = TfidfVectorizer()

vectorizer.fit(corpus)

mykeys = list(vectorizer.vocabulary_.keys())
mykeys.sort()
sorted_dict = {i : vectorizer.vocabulary_[i] for i in mykeys}
```

```
print("Vocabulary:" , sorted_dict)

print(vectorizer.idf_)


vector = vectorizer.transform([corpus[0]])

print()


print(vector.shape)

print()

print("Encoded corpus is:")

print(vector.toarray())
```

OUTPUT: -

```
Vocabulary: {'and': 0, 'be': 1, 'can': 2, 'confusing': 3, 'get': 4, 'hard': 5, 'in': 6, 'it': 7, 'life': 8, 'of': 9, 'poetry': 10, 'sometimes': 11, 'such': 12, 'the':
 'to': 15, 'turn': 16, 'useful': 17, 'wisdom': 18}
[1.40546511 1.40546511 1.         1.40546511 1.40546511 1.40546511
 1.40546511 1.40546511 1.40546511 1.40546511 1.40546511 1.40546511
 1.40546511 1.40546511 1.40546511 1.40546511 1.40546511 1.40546511
 1.40546511]

(1, 19)

Encoded corpus is:
[[0.39204401 0.         0.27894255 0.39204401 0.39204401 0.39204401
  0.         0.         0.39204401 0.         0.         0.39204401
  0.         0.         0.         0.         0.         0.
  0.        ]]
PS C:\Users\saiki\OneDrive\Desktop\7th sem\LABS>
```

PRACTICAL – 4

AIM: - Implementation to Conveít the text to unique integeís withScikitLeaín(HashingVectoíizeí).

Hashing Vectorizer:

Hashing vectorizer is a vectorizer which uses the hashing trick to findthe token string name to feature integer index mapping. Conversion of text documents into matrix is done by this vectorizer where it turns the collection of documents into a sparse matrix which are holding the token occurence counts.

CODE: -

```
from sklearn.feature_extraction.text import

HashingVectorizercorpus = ["Sometimes life an

get confusing and hard",

        "In such times it can be useful to turn to the  wisdom of poetry."]


vectorizer = HashingVectorizer(n_features = 3**5)


vector =

vectorizer.fit_transform(c

orpus)print(vector.shape)
```

OUTPUT: -

```
(2, 243)
PS C:\Users\saiki\OneDrive\Desktop\7th sem\LABS>
```

PRACTICAL – 5

AIM: - Use the Keras deep learning library and split words with (text_to_word_sequence).

KERAS: -

Keras is an open-source deep learning library written in Python. It provides a user-friendly and modular interface for designing, building, training, and deploying various types of artificial neural networks, particularly deep neural networks. Keras was developed with a focus on enabling rapid experimentation and prototyping of deep learning models.

The features of the keras are User-Friendly, Modularity, Compatibility, Flexibility, Extensibility, Visualization.

For installation of the keras in colab we have to use the command

!pip install -q keras

CODE:

```
from keras.preprocessing.text import text_to_word_sequence

txt = "todays hardwork is tomorrows life"

result = text_to_word_sequence(txt)

print(result)
```

OUTPUT: -

```
['todays', 'hardwork', 'is', 'tomorrows', 'life']
PS C:\Users\saiki\OneDrive\Desktop\7th sem\LABS>
```

CODE: -

```
from keras.preprocessing.text import text_to_word_sequence

txt = "todays hardwork is tomorrows life"

words = set(text_to_word_sequence(txt))

vocab_size  =  len(words)

print(vocab_size)
```

OUTPUT: -

```
5
PS C:\Users\saiki\OneDrive\Desktop\7th sem\LABS>
```

PRACTICAL – 6

AIM: - Use the Keras deep learning library and write a code for encoding with(one_hot).

ONE_HOT: -

One-hot encoding is a technique used in deep learning and natural language processing to represent categorical data, such as words or labels, as binary vectors. In one-hot encoding, each category is represented by a vector where all elements are set to zero except for the element corresponding to the category's index, which is set to one. This creates a unique binary representation for each category, allowing them to be easily fed into machine learning algorithms.

For example, consider a simple vocabulary of three words: "apple", and "banana”.

To one-hot encode these words:

"apple" could be represented as [1, 0, 0]

"banana" could be represented as [0, 1, 0]

CODE:

```
from keras.utils import to_categorical

color_labels = [0, 1, 2, 1, 0, 2]

one_hot_encoded = to_categorical(color_labels)

print(one_hot_encoded)
```

OUTPUT: -

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]
 [0. 1. 0.]
 [1. 0. 0.]
 [0. 0. 1.]]
PS C:\Users\saiki\OneDrive\Desktop\7th sem\LABS>
```

PRACTICAL – 7

AIM: - Use the Keras deep learning library and write a code for Hash Encoding with (hashing_trick).

HASH ENCODING: -

Hashing Encoding, often implemented using the **"hashing_trick"** function in Keras, is a technique used to convert categorical data into numerical representations. This is particularly useful when dealing with a large number of categories and you want to reduce the dimensionality of the representation. Let's go through an example using movie genres.

Imagine you have a dataset of movies, and each movie is associated with one or more genres. The genres include "Action", "Comedy", "Drama", "Horror","Romance", and so on. You want to represent these genres numerically using Hashing Encoding.

Before Hashing:

```
from keras.preprocessing.text import text_to_word_sequence
text = " If you don't sacrifice for what you want then what you want is your sacrifice"
result = text_to_word_sequence(text)
print(result)
words = set(text_to_word_sequence(text))
vocab_size = len(words)
print(vocab_size)
```

CODE:

```
['if', 'you', 'dont', 'sacrifice', 'for', 'what', 'you', 'want', 'then', 'what', 'yuou', 'want', 'is', 'your', 'sacrifice']
11
PS C:\Users\saiki\OneDrive\Desktop\7th sem\LABS>
```

BY USING ONE_HOT: -

CODE:

```
from keras.preprocessing.text import one_hot

from keras.preprocessing.text import text_to_word_sequence

text = " If you don’t sacrifice for what you want then what you want is your sacrifice”

words = set(text_to_word_sequence(text))

vocab_size = len(words)

print(vocab_size)

result = one_hot(text, round(vocab_size*1.3))

print(result)
```

OUTPUT:

```
11
[4, 5, 2, 5, 12, 4, 5, 13, 11, 4, 2, 13, 13, 12, 5]
PS C:\Users\saiki\OneDrive\Desktop\7th sem\LABS>
```

AFTER HASHING: -

CODE:

```
from keras.preprocessing.text import hashing_trick

from keras.preprocessing.text import text_to_word_sequence

text = " If you don’t sacrifice for what you want then what you want is your sacrifice”

words = set(text_to_word_sequence(text))

vocab_size = len(words)

print(vocab_size)

result = hashing_trick(text, round(vocab_size*1.3), hash_function='md5')

print(result)
```

OUTPUT: -

```
10
[4, 2, 8, 5, 4, 7, 2, 11, 11, 7, 2, 11, 6, 1, 5]
PS C:\Users\saiki\OneDrive\Desktop\7th sem\LABS>
```

PRACTICAL – 8

AIM : Use the Keras deep learning library give a demo of TokenizerAPI.

1. Tokenization: The tokenizer first breaks down the text into individualwords or tokens. This is done using the fit_on_texts method.
2. Indexing: Each unique word is then assigned a unique integer index. This is done by the word_index property of the tokenizerobject.
3. Text to sequence conversion: The texts_to_sequences method canbe used to convert a list of text samples to a list of sequences of integers.

CODE:

```
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
tokenizer = Tokenizer(num_words=1000)
texts = ["This is a text.", "This is another text."]
tokenizer.fit_on_texts(texts)
tokens = tokenizer.texts_to_sequences(texts)
print(tokens)
```

OUTPUT:

```
[[1, 2, 4, 3], [1, 2, 5, 3]]
PS C:\Users\saiki\OneDrive\Desktop\7th sem\LABS>
```