PRACTICAL NO 1

Name :- Samarth Kuwar
Roll no. :- 06
Batch :- B1
Class :- BECSD

Practical Title :- Convert the text into tokens. Find the word frequency.

Tokenization and Word Frequency (Task #1): Split text into tokens and count word frequencies. For example, using spaCy or NLTK you can tokenize a sentence and then use collections.Counter or NLTK's FreqDist to tally words. The spaCy tutorial shows exactly how to tokenize text and perform frequency analysis realpython.com . (This is trivial and code readily exists; e.g. tokens = word_tokenize(text) and Counter(tokens).)

```
import nltk
nltk.download('punkt')
nltk.download('punkt_tab')
```

```
Output :-
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.

True
```

```
sentence = """This tokenizer divides a text into a list of sentences by
using an unsupervised algorithm to build a model for abbreviation
words, collocations, and words that start sentences. It must be trained
on a large collection of plaintext in the target language before it can
be used."""
tokens = nltk.word_tokenize(sentence)
print("\n")
print(tokens)
from collections import Counter
word_freq = Counter(tokens)
print(word_freq)
```

Output :-

```
['This', 'tokenizer', 'divides', 'a', 'text', 'into', 'a', 'list',
'of', 'sentences', 'by', 'using', 'an', 'unsupervised', 'algorithm',
'to', 'build', 'a', 'model', 'for', 'abbreviation', 'words', ',',
'collocations', ',', 'and', 'words', 'that', 'start', 'sentences', '.',
'It', 'must', 'be', 'trained', 'on', 'a', 'large', 'collection', 'of',
'plaintext', 'in', 'the', 'target', 'language', 'before', 'it', 'can',
'be', 'used', '.']
Counter({'a': 4, 'of': 2, 'sentences': 2, 'words': 2, ',': 2, '.': 2,
'be': 2, 'This': 1, 'tokenizer': 1, 'divides': 1, 'text': 1, 'into': 1,
'list': 1, 'by': 1, 'using': 1, 'an': 1, 'unsupervised': 1,
'algorithm': 1, 'to': 1, 'build': 1, 'model': 1, 'for': 1,
'abbreviation': 1, 'collocations': 1, 'and': 1, 'that': 1, 'start': 1,
'It': 1, 'must': 1, 'trained': 1, 'on': 1, 'large': 1, 'collection': 1,
'plaintext': 1, 'in': 1, 'the': 1, 'target': 1, 'language': 1,
'before': 1, 'it': 1, 'can': 1, 'used': 1})
```

PRACTICAL NO 2

Name :- Samarth Kuwar
Roll no. :- 06
Batch :- B1
Class :- BECSD

Practical Title :- Find the synonym /antonym of a word using WordNet.

Synonyms/Antonyms via WordNet (Task #2): Use NLTK's WordNet interface to find synonyms and antonyms. The approach is to call wordnet.synsets(word), then collect each synset's lemmas for synonyms and use .antonyms() on each lemma for antonyms. For instance, the example shows iterating over synsets("good") and using lemma.name() and lemma.antonyms() to build synonym/antonym lists

```
import nltk
nltk.download('wordnet')
from nltk.corpus import wordnet
synonyms = []
antonyms = []

for syn in wordnet.synsets("good"):
    for l in syn.lemmas():
        synonyms.append(l.name())
        if l.antonyms():
            antonyms.append(l.antonyms()[0].name())

print(set(synonyms))
print(set(antonyms))
```

Output:-

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
{'beneficial', 'estimable', 'adept', 'skilful', 'dependable', 'right',
'salutary', 'soundly', 'sound', 'practiced', 'full', 'just',
'honorable', 'ripe', 'well', 'unspoiled', 'expert', 'good',
'trade_good', 'serious', 'skillful', 'honest', 'proficient', 'near',
'secure', 'in_effect', 'unspoilt', 'upright', 'goodness', 'thoroughly',
'in_force', 'undecomposed', 'safe', 'respectable', 'effective',
'commodity', 'dear'}
{'evil', 'ill', 'bad', 'badness', 'evilness'}
```

PRACTICAL NO 3

Name :- Samarth Kuwar
Roll no. :- 06
Batch :- B1
Class :- BECSD

Practical Title :- Demonstrate a bigram / trigram language model. Generate regular expressions for a given text.

Bigram/Trigram Language Model (Task #3): Build an n-gram model by generating bigrams and trigrams from tokenized text. After tokenizing (with NLTK or similar), you can use nltk.bigrams(tokens) (and nltk.trigrams) to form word pairs/triples. (The frequencies of these n-grams then form a simple language model.) This code is standard in tutorials, and bigram/trigram generation is straightforward with NLTK or Python loops.

```
import nltk
nltk.download('punkt')
from nltk.tokenize import word_tokenize
from nltk.util import bigrams

text = "You are learning from MET Institute of Technology"

tokens = word_tokenize(text)

bigram_list = list(bigrams(tokens))

for bigram in bigram_list:
    print(bigram)
```

Output:-

```
('You', 'are')
('are', 'learning')
('learning', 'from')
('from', 'MET')
('MET', 'Institute')
('Institute', 'of')
('of', 'Technology')
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

PRACTICAL NO 4

Name :- Samarth Kuwar
Roll no. :- 06
Batch :- B1
Class :- BECSD

Practical Title :- Perform Lemmatization and Stemming. Identify parts-of Speech using Penn Treebank tag set.

Lemmatization, Stemming and POS Tagging (Task #4): Apply normalization and part-of-speech tagging. Use NLTK's WordNetLemmatizer (which uses POS) and a stemmer (e.g. Porter) to reduce words. As explained by GeeksforGeeks, lemmatization uses word meaning and POS to get a valid lemma (e.g. "better"→"good"), making it more accurate than blind stemming geeksforgeeks.org . Then use NLTK's pos_tag(tokens) to label each word with Penn Treebank tags (NN, JJ, VBZ, etc.). NLTK's built-in tagger uses the Penn tagset by default (e.g. it tags "John" as 'NNP', "idea" as 'NN', "bad" as 'JJ') nltk.org . Code examples for both lemmatization and POS-tagging are abundant in NLTK tutorials.

```
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

text = "The cats were running faster than the dogs."

tokens = word_tokenize(text)

lemmatized_words = [lemmatizer.lemmatize(word) for word in tokens]

print(f"Original Text: {text}")
print(f"Lemmatized Words: {lemmatized_words}")
```

Output:-

```
Original Text: The cats were running faster than the dogs.
Lemmatized Words: ['The', 'cat', 'were', 'running', 'faster', 'than',
'the', 'dog', '.']
```

```
from nltk.tokenize import word_tokenize
from nltk import pos_tag
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

sentence = "The children are running towards a better place."

tokens = word_tokenize(sentence)

tagged_tokens = pos_tag(tokens)

def get_wordnet_pos(tag):
    if tag.startswith('J'):
        return 'a'
    elif tag.startswith('V'):
        return 'v'
    elif tag.startswith('N'):
        return 'n'
    elif tag.startswith('R'):
        return 'r'
    else:
        return 'n'

lemmatized_sentence = []

for word, tag in tagged_tokens:
    if word.lower() == 'are' or word.lower() in ['is', 'am']:
        lemmatized_sentence.append(word)
    else:
        lemmatized_sentence.append(lemmatizer.lemmatize(word,
get_wordnet_pos(tag)))

print("Original Sentence: ", sentence)
print("Lemmatized Sentence: ", ' '.join(lemmatized_sentence))
```

Output:-

```
Original Sentence:  The children are running towards a better place.
Lemmatized Sentence:  The child are run towards a good place .
```

PRACTICAL NO 5

Name :- Samarth Kuwar
Roll no. :- 06
Batch :- B1
Class :- BECSD

Practical Title :- Implement HMM for POS tagging. Build a Chunker

HMM POS-Tagger and Chunking (Task #5): Build an HMM-based POS tagger and a phrase chunker. A Hidden Markov Model for POS tagging treats tags as hidden states and words as observations; transition and emission probabilities are estimated from tagged data baeldung.com . NLTK even provides HiddenMarkovModelTagger to train on a tagged corpus. For chunking (shallow parsing), one common approach is NLTK's RegexpParser: define regex rules (like <NN.>+ for noun phrases) to group POS-tagged tokens into chunks. For example, you can pos_tag a sentence then apply RegexpParser(r'NP: {+}')* to extract noun chunks dataknowsall.com . (Tutorials on NLTK show exactly this method.)*

```
import spacy

nlp = spacy.load("en_core_web_sm")
doc = nlp("Five Words in Orange Neon")
for chunk in doc.noun_chunks:
    print(chunk.text)
```

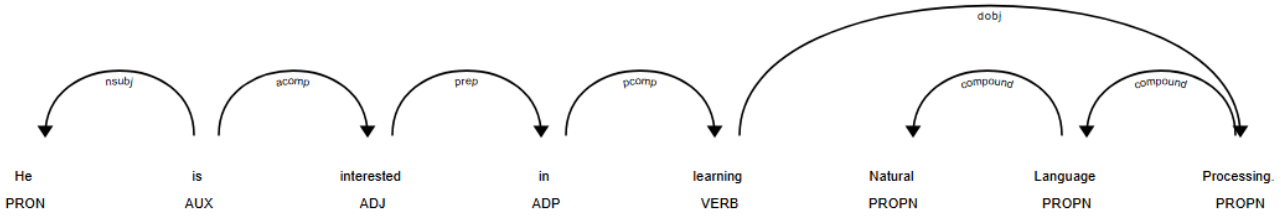
```
Output:-

Five Words
Orange Neon
```

```
import spacy
from spacy import displacy
nlp = spacy.load("en_core_web_sm")

about_interest_text = (
     "He is interested in learning Natural Language Processing."
 )
about_interest_doc = nlp(about_interest_text)
displacy.serve(about_interest_doc, style="dep")
```

Output:-


nsubj
acomp
prep
pcomp
dobj
compound
compound
He
PRON
is
AUX
interested
ADJ
in
ADP
learning
VERB
Natural
PROPN
Language
PROPN
Processing.
PROPN

PRACTICAL NO 6

Name :- Samarth Kuwar
Roll no. :- 06
Batch :- B1
Class :- BECSD

Practical Title :- Implement Named Entity Recognizer.

Named Entity Recognition (Task #6): Identify named entities (people, locations, organizations, etc.) in text. SpaCy (or NLTK's pre-trained taggers) can do this out of the box. As one tutorial notes, NER "locates named entities in unstructured text and classifies them into categories (PERSON, ORG, GPE, etc.)" realpython.com . In spaCy, for instance, after doc = nlp(text), the property doc.ents yields all entities with labels. There are many examples online (e.g. RealPython's guide) showing how to call spaCy or Stanford NER in a few lines.

```
import spacy
nlp = spacy.load("en_core_web_sm")

piano_class_text = (
     "Great Piano Academy is situated"
     " in Mayfair or the City of London and has"
     " world-class piano instructors."
 )
piano_class_doc = nlp(piano_class_text)

for ent in piano_class_doc.ents:
    print(
        f"""
 {ent.text = }
 {ent.start_char = }
 {ent.end_char = }
 {ent.label_ = }
 spacy.explain('{ent.label_}') = {spacy.explain(ent.label_)}"""
 )
```

Output:-

```
ent.text = 'Great Piano Academy'
ent.start_char = 0
ent.end_char = 19
ent.label_ = 'ORG'
spacy.explain('ORG') = Companies, agencies, institutions, etc.

ent.text = 'Mayfair'
ent.start_char = 35
ent.end_char = 42
ent.label_ = 'LOC'
spacy.explain('LOC') = Non-GPE locations, mountain ranges, bodies of
water

ent.text = 'the City of London'
ent.start_char = 46
ent.end_char = 64
ent.label_ = 'GPE'
spacy.explain('GPE') = Countries, cities, states
```

PRACTICAL NO 7

Name :- Samarth Kuwar
Roll no. :- 06
Batch :- B1
Class :- BECSD

Practical Title :- Implement text classifier using logistic regression model

Text Classification with Logistic Regression (Task #8): Build a text classifier using a simple ML model. For example, convert text to feature vectors (Bag-of-Words or TF-IDF) and train a logistic regression with scikit-learn. This is a classic beginner task (e.g. spam detection or topic labeling). As GeeksforGeeks describes, "text classification is fundamental" and can be done by vectorizing text and applying logistic regression geeksforgeeks.org . They even walk through an SMS spam example with CountVectorizer + LogisticRegression geeksforgeeks.org . In practice, students can follow that code or use their own dataset.

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix

#Load and Prepare the Data
data = pd.read_csv('/content/spam.csv', encoding='latin-1')
data.rename(columns={'v1': 'label', 'v2': 'text'}, inplace=True)
data['label'] = data['label'].map({'ham': 0, 'spam': 1})

#Convert text data into a numeric format using CountVectorizer, which
transforms the text into a sparse matrix of token counts.
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(data['text'])
y = data['label']

#Divide the dataset into training and testing sets to evaluate the
model's performance on unseen data.
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25, random_state=42)

#Create and train the logistic regression model using the training set.
model = LogisticRegression(random_state=42)
model.fit(X_train, y_train)
```

```
y_pred = model.predict(X_test)
print("Accuracy_score" ,accuracy_score(y_test, y_pred))

cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(f"[[{cm[0,0]} {cm[0,1]}]")
print(f" [{cm[1,0]} {cm[1,1]}]]")
```

Output:-

```
Accuracy_score 0.9748743718592965
Confusion Matrix:
[[1199 3]
 [32 159]]
```

```
def classify_message(model, vectorizer, message):
    message_vect = vectorizer.transform([message])
    prediction = model.predict(message_vect)
    return "spam" if prediction[0] == 0 else "ham"

message = "Congratulations! You've won a free ticket to Bahamas!"
print(classify_message(model, vectorizer, message))
```

Output:-

```
spam
```

PRACTICAL NO 8

Name :- Samarth Kuwar
Roll no. :- 06
Batch :- B1
Class :- BECSD

Practical Title :- Implement a movie reviews sentiment classifier

Movie Review Sentiment Classifier (Task #9): Classify movie reviews as positive/negative. This is a specialized text classification example. Many public datasets (like the NLTK movie_reviews corpus) exist for this. You can vectorize the review texts and train any classifier (Naive Bayes or logistic). For instance, the CodezUp tutorial shows a complete pipeline: load reviews, TF-IDF vectorize them, and train a Naive Bayes model for sentiment codezup.com . (Open-source code for sentiment analysis abounds, and one can adapt standard examples.)

```
import pandas as pd

df = pd.read_csv(
    "/content/drive/MyDrive/2024-25/1_Odd Sem/9_Teaching Notes/NLP Lab Manual/IMDB Dataset.csv",
    engine="python",   # fallback parser that tolerates messy quotes
    on_bad_lines="skip"  # skip rows that cause parsing errors
)
print(df.shape)
print(df.columns)
df.head()
```

```
Output:-
(50000, 2)
Index(['review', 'sentiment'], dtype='object')
```

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive

```
# Step 2: Import libraries
import pandas as pd
import re
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer

# Step 2: Download NLTK resources
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')

# Step 3: Load dataset (adjust the path if different)
df = pd.read_csv("/content/drive/MyDrive/2024-25/1_Odd Sem/9_Teaching
Notes/NLP Lab Manual/IMDB Dataset.csv")

# Step 4: Check first few rows
print(df.head())

# Step 5: Define text cleaning function
def clean_text(text):
    text = re.sub(r'<.*?>', '', text)  # Remove HTML tags
    text = re.sub(r'[^a-zA-Z]', ' ', text)  # Remove special
characters/numbers
    text = text.lower()  # Convert to lowercase
    words = word_tokenize(text)  # Tokenization
    words = [word for word in words if word not in
stopwords.words('english')]  # Remove stopwords
    lemmatizer = WordNetLemmatizer()
    words = [lemmatizer.lemmatize(word) for word in words]  #
Lemmatization
    return " ".join(words)

# Step 6: Apply cleaning
df['cleaned_review'] = df['review'].apply(clean_text)

# Step 7: Show cleaned data
print(df[['review', 'cleaned_review']].head())

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.metrics import accuracy_score, classification_report

# Convert text data into numerical form
vectorizer = TfidfVectorizer(max_features=5000)  # Use top 5000 words
X = vectorizer.fit_transform(df['cleaned_review'])

# Labels (map positive→1, negative→0)
y = df['sentiment'].map({'positive': 1, 'negative': 0})

# Split data into training and testing
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# Train the model
model = LogisticRegression(max_iter=200)  # increase iterations for
convergence
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)

# Evaluate
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test,
y_pred))
```

Output :-

```
Accuracy: 0.8909
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.88      0.89      5000
           1       0.88      0.90      0.89      5000

    accuracy                           0.89     10000
   macro avg       0.89      0.89      0.89     10000
weighted avg       0.89      0.89      0.89     10000
```

```
# Test with a custom review
test_review = ["The movie was absolutely not amazing"]
test_review_tfidf = vectorizer.transform(test_review)
prediction = model.predict(test_review_tfidf)
print("Sentiment:", "Positive" if prediction[0] == 1 else "Negative")
```

```
Output:-
Sentiment: Positive
```

```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer

# Download lexicon (only needed once)
nltk.download('vader_lexicon')

# Initialize VADER
vader = SentimentIntensityAnalyzer()

test_review = "The movie was absolutely not amazing"
vader_score = vader.polarity_scores(test_review)

print("VADER Output:", vader_score)

if vader_score['compound'] >= 0.05:
    print("VADER Sentiment: Positive")
elif vader_score['compound'] <= -0.05:
    print("VADER Sentiment: Negative")
else:
    print("VADER Sentiment: Neutral")
```

```
Output:-
VADER Output: {'neg': 0.401, 'neu': 0.599, 'pos': 0.0, 'compound':
-0.5188}
VADER Sentiment: Negative
```

PRACTICAL NO 9

Name :- Samarth Kuwar
Roll no. :- 06
Batch :- B1
Class :- BECSD

Practical Title :- Implement RNN for sequence labelling

RNN for Sequence Labeling (Task #10): Implement a recurrent neural network to label sequences (e.g. NER or POS). This is more advanced but still supported by many tutorials. RNNs (especially LSTM/GRU variants) are well-known to excel at sequence tasks. GeeksforGeeks notes that RNNs "have proven to be highly effective in NLP tasks, particularly in sequence labeling" (including POS tagging and NER) geeksforgeeks.org . In practice, one can use Keras/TensorFlow to build a simple RNN or LSTM model: for example, a Keras Embedding + LSTM + TimeDistributed(Dense) network to tag each word. Ready-made example code is available (see TensorFlow/Keras guides or GeeksforGeeks examples of RNN tagging geeksforgeeks.org)

```
import numpy as np
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.optimizers import Adam

# --------------------------
# 1. Load IMDb dataset
# --------------------------
vocab_size = 10000   # top 10k words
max_sequence_length = 200  # pad / truncate all reviews to 200 tokens

(X_train, y_train), (X_test, y_test) =
imdb.load_data(num_words=vocab_size)

# Pad sequences to fixed length
X_train = pad_sequences(X_train, maxlen=max_sequence_length,
padding='post')
X_test  = pad_sequences(X_test,  maxlen=max_sequence_length,
padding='post')

print("Training data shape:", X_train.shape)
print("Test data shape:", X_test.shape)
```

```
# --------------------------
# 2. Build the model
# --------------------------
embedding_dim = 128
lstm_units = 64
num_labels = 1   # binary classification (positive/negative)

model = Sequential()
model.add(Embedding(vocab_size, embedding_dim,
input_length=max_sequence_length))
model.add(LSTM(lstm_units))
model.add(Dense(num_labels, activation='sigmoid'))   # sigmoid for
binary sentiment

model.compile(loss='binary_crossentropy', optimizer=Adam(1e-3),
metrics=['accuracy'])
model.summary()

# --------------------------
# 3. Train the model
# --------------------------
history = model.fit(
    X_train, y_train,
    epochs=2,   # keep epochs small for quick demo
    batch_size=128,
    validation_split=0.2,
    verbose=1
)

# --------------------------
# 4. Evaluate on test set
# --------------------------
loss, acc = model.evaluate(X_test, y_test, verbose=0)
print(f"\nTest Accuracy: {acc:.4f}")

# --------------------------
# 5. Try a few sample predictions
# --------------------------
word_index = imdb.get_word_index()
reverse_word_index = {value: key for (key, value) in
word_index.items()}

def decode_review(sequence):
```

```
    # Convert word indices back to words
    return " ".join([reverse_word_index.get(i - 3, "?") for i in
sequence])

# Pick 3 random test samples
for i in np.random.randint(0, len(X_test), 3):
    review = decode_review(X_test[i])
    pred = model.predict(X_test[i].reshape(1, -1), verbose=0)[0][0]
    print("\nReview:", review[:300], "...")
    print("Actual Label:", "Positive" if y_test[i] == 1 else
"Negative")
    print("Predicted Probability:", pred)
    print("Predicted Label:", "Positive" if pred > 0.5 else "Negative")
```

Output:-

Data

- Train shape: `(25000, 200)`
- Test shape: `(25000, 200)`

Keras note

- `Embedding(input_length=...)` is deprecated (warning shown).

Training (2 epochs)

- Epoch 1 — loss **0.6856**, acc **0.5473**; val_loss **0.7020**, val_acc **0.5784**
- Epoch 2 — loss **0.6559**, acc **0.5839**; val_loss **0.5881**, val_acc **0.6384**

Final test accuracy: 0.6371

Sample predictions

1. Review 1 — Actual: **Negative** → Pred prob **0.5098423**, Pred label **Positive** (**incorrect**)

2. Review 2 — Actual: **Negative** → Pred prob **0.5098672**, Pred label **Positive** (**incorrect**)

3. Review 3 — Actual: **Positive** → Pred prob **0.509843**, Pred label **Positive** (**correct**)

PRACTICAL NO 10

Name :- Samarth Kuwar
Roll no. :- 06
Batch :- B1
Class :- BECSD

Practical Title :- Implementation of LSTM-based Part-of-Speech (POS) Tagging using Keras.

LSTM-based POS Tagging (Task #11): Train an LSTM network specifically for POS tagging. An LSTM is a type of RNN that handles long-range dependencies. As the literature notes, "LSTMs can capture long-term dependencies in sequential data" making them ideal for language tasks geeksforgeeks.org . You would prepare input sequences (e.g. padded word indices) and output tag sequences, then train a Keras LSTM model (often bidirectional) to predict tags. Example projects and GitHub repos (e.g. Bidirectional LSTM taggers) are readily found. Although deeper than earlier tasks, code and tutorials exist for LSTM taggers (the cited RNN guide also discusses LSTM units geeksforgeeks.org).

```
# --------------------------
# Task 10: RNN for Sequence Labeling
# --------------------------
!pip install datasets tensorflow --quiet

import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, TimeDistributed,
Dense
from tensorflow.keras.preprocessing.sequence import pad_sequences
from datasets import load_dataset

# --------------------------
# 1. Load dataset (CoNLL-2003 NER)
# --------------------------
dataset = load_dataset("xtreme", "PAN-X.en")


# We'll use the training split
train_data = dataset["train"]
```

```
val_data   = dataset["validation"]

# Word and label vocabularies
words = list(set(w for sent in train_data["tokens"] for w in sent))
labels = list(set(l for sent in train_data["ner_tags"] for l in sent))

word2idx = {w: i+2 for i, w in enumerate(words)}  # reserve 0=PAD,
1=UNK
word2idx["<PAD>"] = 0
word2idx["<UNK>"] = 1

idx2word = {i: w for w, i in word2idx.items()}

label2idx = {l: i for i, l in enumerate(sorted(set(labels)))}
idx2label = {i: l for l, i in label2idx.items()}

print("Vocab size:", len(word2idx))
print("Number of labels:", len(label2idx))

# --------------------------
# 2. Preprocess (convert words/labels to indices + pad)
# --------------------------
MAX_LEN = 50  # limit sentence length
def encode(sent, tag_seq):
    # words to indices
    x = [word2idx.get(w, word2idx["<UNK>"]) for w in sent]
    # labels to indices
    y = [label2idx[t] for t in tag_seq]
    return x, y

X_train, y_train = [], []
for s, t in zip(train_data["tokens"], train_data["ner_tags"]):
    x, y = encode(s, t)
    X_train.append(x)
    y_train.append(y)

X_val, y_val = [], []
for s, t in zip(val_data["tokens"], val_data["ner_tags"]):
    x, y = encode(s, t)
    X_val.append(x)
    y_val.append(y)

# Pad sequences
```

```
X_train = pad_sequences(X_train, maxlen=MAX_LEN, padding="post")
X_val   = pad_sequences(X_val,   maxlen=MAX_LEN, padding="post")
y_train = pad_sequences(y_train, maxlen=MAX_LEN, padding="post")
y_val   = pad_sequences(y_val,   maxlen=MAX_LEN, padding="post")

# Convert labels to categorical (one-hot)
y_train = tf.keras.utils.to_categorical(y_train,
num_classes=len(label2idx))
y_val   = tf.keras.utils.to_categorical(y_val,
num_classes=len(label2idx))

print("Train shape:", X_train.shape, y_train.shape)

# --------------------------
# 3. Build RNN model
# --------------------------
model = Sequential()
model.add(Embedding(len(word2idx), 128, input_length=MAX_LEN,
mask_zero=True))
model.add(LSTM(64, return_sequences=True))
model.add(TimeDistributed(Dense(len(label2idx), activation="softmax")))

model.compile(optimizer="adam", loss="categorical_crossentropy",
metrics=["accuracy"])
model.summary()

# --------------------------
# 4. Train model (few epochs for demo)
# --------------------------
history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=2,
    batch_size=64
)

# --------------------------
# 5. Test on sample sentence
# --------------------------
i = np.random.randint(0, len(X_val))
sample = X_val[i].reshape(1, -1)
pred = model.predict(sample)
```

```
pred_labels = [idx2label[np.argmax(p)] for p in pred[0]]
input_tokens = [idx2word.get(idx, "") for idx in X_val[i] if idx != 0]

print("\nInput Sentence:", " ".join(input_tokens))
print("Predicted Labels:", pred_labels[:len(input_tokens)])
```

Output:-

Vocab size: 33719
Number of labels: 7
Train shape: (20000, 50) (20000, 50, 7)

Model: "sequential_2"

Layer (type)	Output Shape	Param #
embedding_2 (Embedding)	?	0 (unbuilt)
lstm_2 (LSTM)	?	0 (unbuilt)
time_distributed_1 (TimeDistributed)	?	0 (unbuilt)

Total params: 0 (0.00 B)

Trainable params: 0 (0.00 B)

Non-trainable params: 0 (0.00 B)

Epoch 1/2
313/313 ━━━━━━━━━━━━━━━━━━━━ **55s** 158ms/step - accuracy: 0.7977 - loss: 1.3509 - val_accuracy: 0.5852 - val_loss: 0.7255
Epoch 2/2
313/313 ━━━━━━━━━━━━━━━━━━━━ **79s** 150ms/step - accuracy: 0.5815 - loss: 0.5395 - val_accuracy: 0.5723 - val_loss: 0.6244
1/1 ━━━━━━━━━━━━━━━━━━━━ **1s** 748ms/step

Input Sentence: `` Highway Do n't Care ''
Predicted Labels: [0, 4, 4, 4, 4, 0]