SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (DATA SCIENCE)

NAME:DIVYESH KHUNT **SAPID:60009210116** **BATCH:D12**

COURSE CODE: DJ19DSC501 **DATE:10-11-23**

COURSE NAME: Machine Learning - II **CLASS: AY 2023-24**

LAB EXPERIMENT NO.6

AIM / OBJECTIVE:

Implement LSTM Sentiment Analysis on text dataset to evaluate customer reviews.

DESCRIPTION OF EXPERIMENT:

Python sentiment analysis is a methodology for analyzing a piece of text to discover the sentiment hidden within it. It accomplishes this by combining machine learning and natural language processing (NLP). Sentiment analysis allows you to examine the feelings expressed in a piece of text. It is essential for businesses to gauge customer response.

Preprocessing -

1) Normalization - Words which look different due to casing or written another way but are the same in meaning need to be process correctly. Normalisation processes ensure that these words are treated equally. For example, changing numbers to their word equivalents or converting the casing of all the text.
   a) Casing the Characters - Converting character to the same case so the same words are recognised as the same. (all lowercase)
   b) Removing - Stand alone punctuations, special characters and numerical tokens are removed as they do not contribute to sentiment which leaves only alphabetic characters. This step needs the use of tokenized words as they have been split appropriately for us to remove. We need to remove the special characters, numbers from the text. We can use the regular expression operations library of Python.

2) Tokenization - Tokenization is the process of breaking down chunks of text into smaller pieces. It converts text into tokens before transforming it into vectors. It is also easier to filter out unnecessary tokens. spaCy comes with a default processing pipeline that begins with tokenization, making this process a snap. In spaCy, you can do either sentence tokenization or word tokenization:

- Word tokenization breaks text down into individual words.
- Sentence tokenization breaks text down into individual sentences.

3) Stopwords - Stop words are the most commonly occuring words which are not relevant in the context of the data and do not contribute any deeper meaning to the phrase. In this case it contains no sentiment. We need to remove them as part of text preprocessing. nltk has a list of stopwords of every language.

SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)

4) Obtaining the stem words

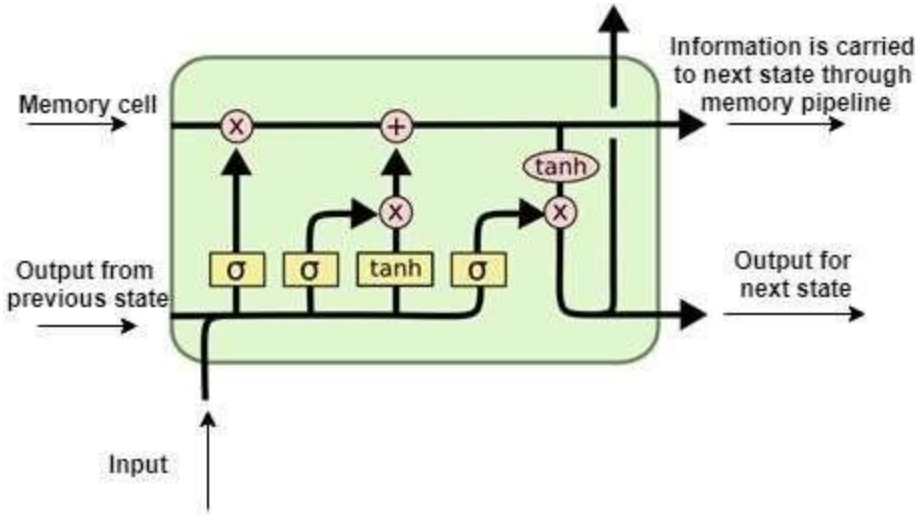
A stem is a part of a word responsible for its lexical meaning. The two popular techniques of obtaining the root/stem words are Stemming and Lemmatization

a) Stemming - Stemming is a technique used to extract the base form of the words by removing affixes from them. It is just like cutting down the branches of a tree to its stems. For example, the stem of the words eating, eats, eaten is eat.

b) Lemmitization - This process finds the base or dictionary form of the word known as the lemma. This is done through the use of vocabulary (dictionary importance of words) and morphological analysis (word structure and grammar relations)

5) Vectorization - use a count vectorizer from the Scikit-learn library to transform the text in data frame into a bag of words model, which will contain a sparse matrix of integers. The number of occurrences of each word will be counted.

Sentiment Analysis using LSTM: Use Keras



Hyperparameters to tune -

1. Layers - Explore additional hierarchical learning capacity by adding more layers and varied numbers of neurons in each layer
2. Number of inputs in dense layer - Dense layers improve overall accuracy and 5–10 units or nodes per layer is a good base
3. Dropout - Slow down learning with regularization methods like dropout on the recurrent LSTM connections.A good starting point is 20% but the dropout value should be kept small (up to 50%). The 20% value is widely accepted as the best compromise between preventing model overfitting and retaining model accuracy.
4. Learning Rate - This hyperparameter defines how quickly the network updates its parameters.
5. Decay Rate - weight decay can be added in the weight update rule that makes the weights decay to zero exponentially, if no other weight update is scheduled. After each update, the weights are multiplied by a factor slightly less than 1, thereby preventing them from growing to huge. This specifies regularization in the network.

6. Number of epochs

Sentiment Analysis using TextBlob:

TextBlob is a Python library for processing textual data. It provides a consistent API for diving into common natural language processing (NLP) tasks such as part-of-speech tagging, noun phrase extraction, sentiment analysis, and more.

The two measures that are used to analyze the sentiment are:

- Polarity – talks about how positive or negative the opinion is
- Subjectivity – talks about how subjective the opinion is

TextBlob(text).sentiment gives us the Polarity, Subjectivity values.

Polarity ranges from -1 to 1 (1 is more positive, 0 is neutral, -1 is more negative)

Subjectivity ranges from 0 to 1(0 being very objective and 1 being very subjective)

```
res = TextBlob("I love horror films").sentiment
res
```

```
Sentiment(polarity=0.5, subjectivity=0.6)
```

Example of TextBlob sentiment

Workflow -

1. Preprocess data.
2. Split data into training and evaluation sets.
3. Select a model architecture.
4. Use training data to train model.
5. Use test data to evaluate the performance of model.

1. **Apply preprocessing techniques and LSTM on the dataset. Show accuracy achieved on the test dataset by providing classification report.**
2. **Perform LSTM hyperparameter tuning to improve accuracy score.**
3. **Show how LSTM model compares to built-in classifier provided by TextBlob.**
4. **State the applications of sentiment analysis**
5. **State the challenges faced while performing sentiment analysis.**

SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense

df = pd.read_csv('/content/Restaurant_Reviews.tsv',sep='\t')

texts =df["Review"].values
labels =df["Liked"].values
```

```
max_words = 10000
max_len = 100

tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(texts)

sequences = tokenizer.texts_to_sequences(texts)
X = pad_sequences(sequences, maxlen=max_len)
y = np.array(labels)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

embedding_dim = 50
lstm_units = 50

model = Sequential()
model.add(Embedding(input_dim=max_words, output_dim=embedding_dim, input_length=max_len))
model.add(LSTM(units=lstm_units))
model.add(Dense(units=1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=5, batch_size=32, validation_split=0.2)

y_pred_prob = model.predict(X_test)
y_pred = (y_pred_prob > 0.5).astype(int)

print(classification_report(y_test, y_pred))
```

SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)

```
Epoch 1/5
20/20 [==============================] - 3s 80ms/step - loss: 0.6919 - accuracy: 0.5234 - val_loss: 0.6905 - val_accuracy: 0.5437
Epoch 2/5
20/20 [==============================] - 1s 54ms/step - loss: 0.6721 - accuracy: 0.7250 - val_loss: 0.6643 - val_accuracy: 0.7063
Epoch 3/5
20/20 [==============================] - 1s 56ms/step - loss: 0.5814 - accuracy: 0.7781 - val_loss: 0.6056 - val_accuracy: 0.6875
Epoch 4/5
20/20 [==============================] - 1s 50ms/step - loss: 0.4308 - accuracy: 0.8281 - val_loss: 0.5409 - val_accuracy: 0.7375
Epoch 5/5
20/20 [==============================] - 1s 49ms/step - loss: 0.2637 - accuracy: 0.9203 - val_loss: 0.4887 - val_accuracy: 0.8000
7/7 [==============================] - 1s 13ms/step
              precision    recall  f1-score   support

           0       0.74      0.67      0.70        96
           1       0.72      0.78      0.75       104

    accuracy                           0.73       200
   macro avg       0.73      0.72      0.72       200
weighted avg       0.73      0.72      0.72       200
```

```python
from tensorflow.keras.callbacks import EarlyStopping

# Hyperparameter tuning
embedding_dims = [50, 100]
lstm_units_vals = [50, 100]
batch_sizes = [32, 64]
epochs = 10

best_accuracy = 0
best_model = None

for embedding_dim in embedding_dims:
    for lstm_units in lstm_units_vals:
        for batch_size in batch_sizes:
            model = Sequential()
            model.add(Embedding(input_dim=max_words, output_dim=embedding_dim, input_length=max_len))
            model.add(LSTM(units=lstm_units))
            model.add(Dense(units=1, activation='sigmoid'))
            model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
            # Early stopping to prevent overfitting
            early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
            model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size, validation_split=0.2, callbacks=[early_stopping])
            _, accuracy = model.evaluate(X_test, y_test)
            print(f"Embedding_dim={embedding_dim}, LSTM_units={lstm_units}, Batch_size={batch_size}, Accuracy={accuracy}")
            if accuracy > best_accuracy:
                best_accuracy = accuracy
                best_model = model

# Use the best model for predictions
y_pred_prob = best_model.predict(X_test)
y_pred = (y_pred_prob > 0.5).astype(int)

print(classification_report(y_test, y_pred))
```

SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)

```
10/10 [==============================] - 1s 118ms/step - loss: 0.6433 - accuracy: 0.7922 - val_loss: 0.6508 - val_accuracy: 0.6750
Epoch 4/10
10/10 [==============================] - 1s 104ms/step - loss: 0.5482 - accuracy: 0.8094 - val_loss: 0.5946 - val_accuracy: 0.6625
Epoch 5/10
10/10 [==============================] - 1s 118ms/step - loss: 0.4572 - accuracy: 0.8125 - val_loss: 0.5911 - val_accuracy: 0.7437
Epoch 6/10
10/10 [==============================] - 1s 141ms/step - loss: 0.3600 - accuracy: 0.9125 - val_loss: 0.5340 - val_accuracy: 0.7625
Epoch 7/10
10/10 [==============================] - 1s 137ms/step - loss: 0.2628 - accuracy: 0.9469 - val_loss: 0.5189 - val_accuracy: 0.7750
Epoch 8/10
10/10 [==============================] - 1s 95ms/step - loss: 0.1785 - accuracy: 0.9625 - val_loss: 0.4932 - val_accuracy: 0.8250
Epoch 9/10
10/10 [==============================] - 1s 99ms/step - loss: 0.1234 - accuracy: 0.9766 - val_loss: 0.4948 - val_accuracy: 0.8375
Epoch 10/10
10/10 [==============================] - 1s 99ms/step - loss: 0.0896 - accuracy: 0.9844 - val_loss: 0.4882 - val_accuracy: 0.8375
7/7 [==============================] - 0s 26ms/step - loss: 0.5977 - accuracy: 0.7250
Embedding_dim=100, LSTM_units=50, Batch_size=64, Accuracy=0.7250000238418579
Epoch 1/10
20/20 [==============================] - 5s 143ms/step - loss: 0.6900 - accuracy: 0.5281 - val_loss: 0.6832 - val_accuracy: 0.6000
Epoch 2/10
20/20 [==============================] - 2s 103ms/step - loss: 0.6304 - accuracy: 0.7234 - val_loss: 0.6165 - val_accuracy: 0.6562
Epoch 3/10
20/20 [==============================] - 2s 103ms/step - loss: 0.4227 - accuracy: 0.8516 - val_loss: 0.5419 - val_accuracy: 0.7188
Epoch 4/10
20/20 [==============================] - 3s 147ms/step - loss: 0.9100 - accuracy: 0.8516 - val_loss: 0.5103 - val_accuracy: 0.7312
Epoch 5/10
20/20 [==============================] - 2s 113ms/step - loss: 0.1964 - accuracy: 0.9563 - val_loss: 0.4853 - val_accuracy: 0.7875
Epoch 6/10
20/20 [==============================] - 2s 104ms/step - loss: 0.1334 - accuracy: 0.9812 - val_loss: 0.4662 - val_accuracy: 0.8188
Epoch 7/10
20/20 [==============================] - 2s 101ms/step - loss: 0.0782 - accuracy: 0.9875 - val_loss: 0.4535 - val_accuracy: 0.8188
Epoch 8/10
20/20 [==============================] - 2s 103ms/step - loss: 0.0532 - accuracy: 0.9969 - val_loss: 0.4702 - val_accuracy: 0.8125
Epoch 9/10
20/20 [==============================] - 2s 103ms/step - loss: 0.0358 - accuracy: 0.9984 - val_loss: 0.5197 - val_accuracy: 0.8125
Epoch 10/10
20/20 [==============================] - 3s 165ms/step - loss: 0.0252 - accuracy: 0.9984 - val_loss: 0.5656 - val_accuracy: 0.7937
7/7 [==============================] - 1s 28ms/step - loss: 0.5686 - accuracy: 0.7250
Embedding_dim=100, LSTM_units=100, Batch_size=32, Accuracy=0.7250000238418579
   Embedding_dim=100, LSTM_units=100, Batch_size=32, Accuracy=0.7250000238418579
   Epoch 1/10
   10/10 [==============================] - 4s 258ms/step - loss: 0.6922 - accuracy: 0.5141 - val_loss: 0.6884 - val_accuracy: 0.6500
   Epoch 2/10
   10/10 [==============================] - 2s 189ms/step - loss: 0.6748 - accuracy: 0.7469 - val_loss: 0.6720 - val_accuracy: 0.6750
   Epoch 3/10
   10/10 [==============================] - 2s 187ms/step - loss: 0.6079 - accuracy: 0.7437 - val_loss: 0.6432 - val_accuracy: 0.6750
   Epoch 4/10
   10/10 [==============================] - 3s 340ms/step - loss: 0.4676 - accuracy: 0.8250 - val_loss: 0.5576 - val_accuracy: 0.7063
   Epoch 5/10
   10/10 [==============================] - 2s 205ms/step - loss: 0.3501 - accuracy: 0.8813 - val_loss: 0.5514 - val_accuracy: 0.7125
   Epoch 6/10
   10/10 [==============================] - 2s 178ms/step - loss: 0.3087 - accuracy: 0.9125 - val_loss: 0.5334 - val_accuracy: 0.7437
   Epoch 7/10
   10/10 [==============================] - 2s 170ms/step - loss: 0.2233 - accuracy: 0.9391 - val_loss: 0.5505 - val_accuracy: 0.7688
   Epoch 8/10
   10/10 [==============================] - 2s 173ms/step - loss: 0.1425 - accuracy: 0.9547 - val_loss: 0.5584 - val_accuracy: 0.7937
   Epoch 9/10
   10/10 [==============================] - 3s 338ms/step - loss: 0.0937 - accuracy: 0.9844 - val_loss: 0.6789 - val_accuracy: 0.7125
   7/7 [==============================] - 1s 63ms/step - loss: 0.6008 - accuracy: 0.6650
   Embedding_dim=100, LSTM_units=100, Batch_size=64, Accuracy=0.6650000214576721
   7/7 [==============================] - 2s 43ms/step
                 precision    recall  f1-score   support

              0       0.73      0.83      0.78        96
              1       0.82      0.71      0.76       104

       accuracy                           0.77       200
      macro avg       0.77      0.77      0.77       200
   weighted avg       0.78      0.77      0.77       200
```

SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)

```python
from textblob import TextBlob

max_words = 10000
max_len = 100
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)
X = pad_sequences(sequences, maxlen=max_len)
y = np.array(labels)
embedding_dim = 50
lstm_units = 50

model = Sequential()
model.add(Embedding(input_dim=max_words, output_dim=embedding_dim, input_length=max_len))
model.add(LSTM(units=lstm_units))
model.add(Dense(units=1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=5, batch_size=32, validation_split=0.2)

y_pred_prob_lstm = model.predict(X_test)
y_pred_lstm = (y_pred_prob_lstm > 0.5).astype(int)

print("Classification Report for LSTM Model:")
print(classification_report(y_test, y_pred_lstm))

def textblob_classifier(text):
    analysis = TextBlob(str(text))
    return 1 if analysis.sentiment.polarity >= 0 else 0

y_pred_textblob = np.array([textblob_classifier(text) for text in X_test])

print("\nClassification Report for TextBlob:")
print(classification_report(y_test, y_pred_textblob))
```

```
Epoch 1/5
20/20 [==============================] - 4s 82ms/step - loss: 0.6921 - accuracy: 0.4969 - val_loss: 0.6926 - val_accuracy: 0.4875
Epoch 2/5
20/20 [==============================] - 1s 57ms/step - loss: 0.6738 - accuracy: 0.6281 - val_loss: 0.6682 - val_accuracy: 0.6687
Epoch 3/5
20/20 [==============================] - 1s 56ms/step - loss: 0.5829 - accuracy: 0.7500 - val_loss: 0.6163 - val_accuracy: 0.7375
Epoch 4/5
20/20 [==============================] - 1s 52ms/step - loss: 0.4050 - accuracy: 0.8578 - val_loss: 0.6914 - val_accuracy: 0.5750
Epoch 5/5
20/20 [==============================] - 1s 65ms/step - loss: 0.2831 - accuracy: 0.9000 - val_loss: 0.5230 - val_accuracy: 0.7875
7/7 [==============================] - 1s 14ms/step
Classification Report for LSTM Model:
              precision    recall  f1-score   support

           0       0.66      0.71      0.68        96
           1       0.71      0.66      0.69       104

    accuracy                           0.69       200
   macro avg       0.69      0.69      0.68       200
weighted avg       0.69      0.69      0.69       200


Classification Report for TextBlob:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        96
           1       0.52      1.00      0.68       104

    accuracy                           0.52       200
   macro avg       0.26      0.50      0.34       200
weighted avg       0.27      0.52      0.36       200
```