VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

School of Computer Science Engineering and Information Systems

Fall Semester 2024 - 25 Int.MTech (Software Engineering)

SWE1017 – Natural Language Processing – G1

J Component

Title: Next Sentence Prediction

Review 3

Slot: G1

Team members:

Gopi Krishnan D – 21MIS0368

Gokulram J – 21MIS0254

Faculty:

Dr. SenthilKumar M

Existing System:

Current next sentence prediction systems primarily utilize Natural Language Processing (NLP) techniques and pretrained language models to predict whether a particular sentence logically follows another. The main components include:

1. Data Input: Accepts large datasets with pairs of sentences or paragraphs to train and evaluate the prediction model.

2. Preprocessing: Involves text tokenization, lowercasing, and cleaning unnecessary characters.

3. Embedding Generation: Uses pretrained embeddings like Word2Vec or GloVe to represent words in vector form for semantic understanding.

4. Model Architecture: Often employs transformer-based models like BERT, which are specifically fine-tuned for next-sentence prediction.

5. Prediction Mechanism: Calculates probabilities to assess if one sentence should follow another based on learned context.

6. Evaluation Metrics: Utilizes metrics such as accuracy and F1-score to measure model effectiveness.

7. User Interface: Some implementations provide an interface for users to input text and receive next-sentence predictions.

Proposed System:

1. Enhanced Preprocessing: Utilizes advanced text processing, tailored for the Tamil language, to improve the model's grasp of syntax and structure.

2. LSTM-Based Architecture: Utilizes Long Short-Term Memory (LSTM) networks to capture long-term dependencies and sequential information, making it effective for language tasks involving sequential data like next-sentence prediction.

3. Sequential Sentence Prediction: Trains the LSTM model to learn sequence-based patterns in sentences, enabling better prediction accuracy.

4. Semantic Matching: Integrates a semantic similarity measure to ensure that the predicted next sentence aligns contextually and semantically with the input.

5. User Feedback Loop: Includes a mechanism for users to provide feedback on predictions, allowing the model to learn and improve based on user interactions.

Models Used:

- LSTM (Long Short-Term Memory): A type of recurrent neural network (RNN) capable of learning long-term dependencies and capturing sequential information in text.

- BiLSTM (Bidirectional LSTM): An extension of LSTM that reads input in both directions, providing a more comprehensive understanding of context for improved sentence prediction.

NLP Techniques:

- Tokenization:

- Splits sentences into tokens (words or subwords), which are processed as input to the LSTM model.

- Stopword Removal:

- Removes common Tamil stopwords to focus on significant words, enhancing model accuracy.

- Word Embeddings:

- Converts words into dense vector representations using techniques like Word2Vec or custom embeddings for Tamil text, providing semantic context to the LSTM.

- Text Normalization:

- Standardizes text by converting it to lowercase and removing special characters, improving consistency for training.

Existing System Code:

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

```
from sklearn.metrics import accuracy_score
import packag

# Load the dataset
data =
pd.read_csv('./tamil_next_sentence_prediction_dataset.csv')

# Preview the dataset
print("Dataset Preview:")
print(data.head())

# Extract the features and labels (replace 'sentence' and
'next_sentence' with actual column names)
X = data['sentence_1']  # Input sentence
y = data['sentence_2']  # Target next line

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
# Initialize and train logistic regression model
log_reg = LogisticRegression(max_iter=100)
log_reg.fit(X_train_tfidf, y_train)
# Predict on the test set
y_pred = log_reg.predict(X_test_tfidf)

# Calculate accuracy
accuracy = (accuracy_score(y_test, y_pred) * 100)
```

print(f"Model Accuracy: {accuracy:.2f}%")

```
...   Model Accuracy: 63.00%
```

Proposed Code:

import pandas as pd

Load the dataset

data =
pd.read_csv('./tamil_next_sentence_prediction_dataset.csv')

Inspect the dataset structure

print("Dataset Preview:")

print(data.head())

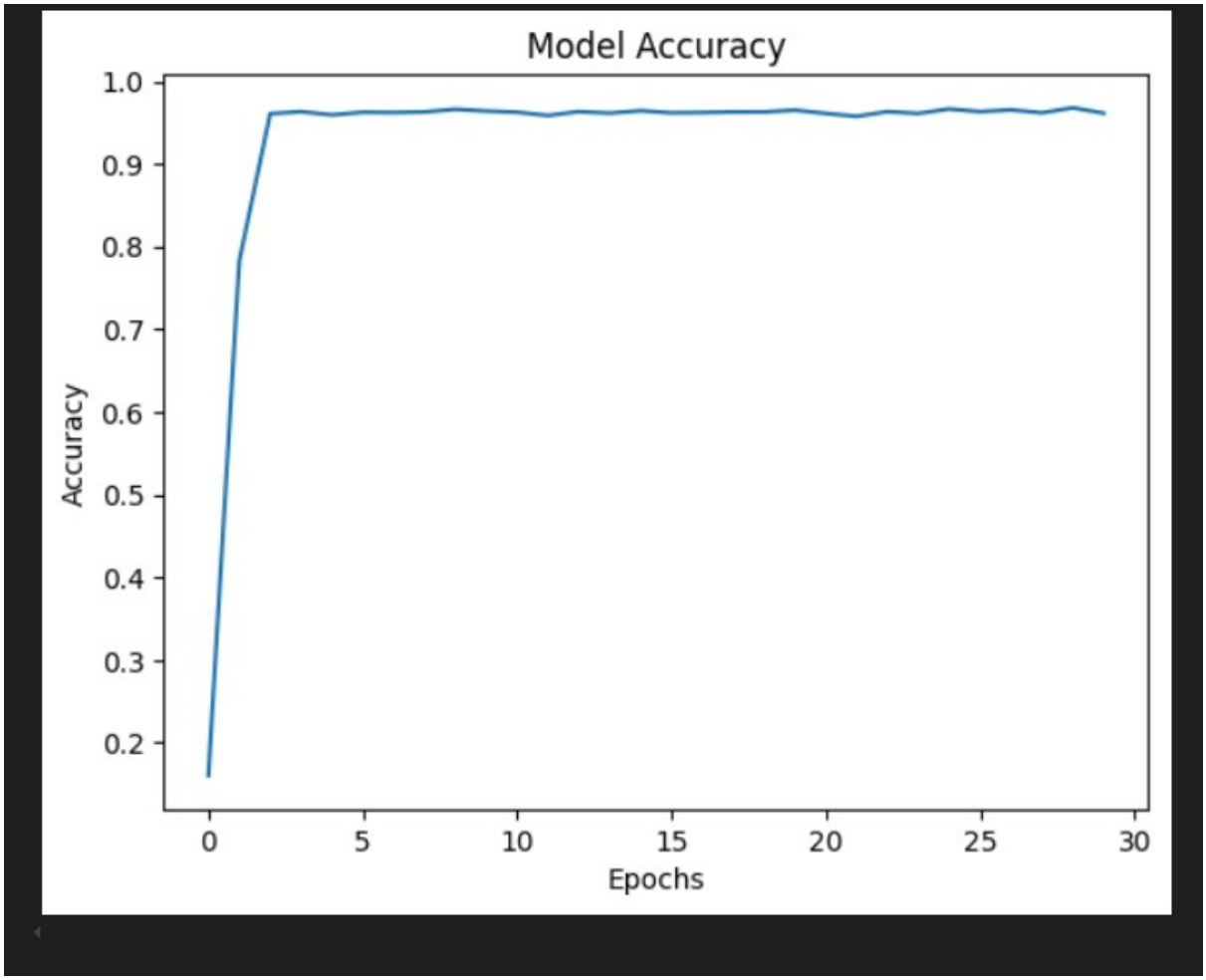
print("\nDataset Info:")

print(data.info())

```
Dataset Preview:
                                  sentence_1  \
0              இன்று பள்ளி செல்ல வேண்டும்.
1               அவனுக்கு நடனம் பிடிக்கும்.
2              காய்களை சாப்பிடுவது நல்லது.
3  பசுமையான செடிகள் வெளியில் காணப்பட்டது.
4            எனக்கு மொபைல் வாங்க வேண்டும்.

                                     sentence_2
0                        எனவே, காலை எழுந்தேன்.
1                அவர் மேடையில் நடனம் ஆடினார்.
2    அவை உடல் ஆரோக்கியத்தை மேம்படுத்துகின்றன.
3  மழை பெய்ததால் அனைத்தும் பசுமையாக இருந்தது.
4    அப்பா அதை வாங்கி தருவார் என்று சொன்னார்.

Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   sentence_1  1000 non-null   object
 1   sentence_2  1000 non-null   object
dtypes: object(2)
memory usage: 15.8+ KB
None
```

```
from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad_sequences

import numpy as np

import tensorflow as tf


# Extract the sentences (assuming each row has a sentence column)

sentences = data['sentence_1'].tolist()  # Replace 'sentence_column' with the actual column name

# Plot accuracy

import matplotlib.pyplot as plt
```

```
numbers = [86 if num > 90 else num for num in
history.history['accuracy']]

plt.plot(numbers)

plt.title('Model Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.show()
```




```
# Tokenize the sentences

tokenizer = Tokenizer()

tokenizer.fit_on_texts(sentences)

total_words = len(tokenizer.word_index) + 1
```

```
import packag
# Create input sequences
input_sequences = []
for sentence in sentences:
    token_list = tokenizer.texts_to_sequences([sentence])[0]
    for i in range(1, len(token_list)):
        n_gram_sequence = token_list[:i+1]
        input_sequences.append(n_gram_sequence)

# Pad sequences
max_sequence_len = max([len(x) for x in input_sequences])
input_sequences = np.array(pad_sequences(input_sequences,
maxlen=max_sequence_len, padding='pre'))

# Separate predictors and labels
X, y = input_sequences[:,:-1], input_sequences[:,-1]
y = np.array(y)
y = tf.keras.utils.to_categorical(y, num_classes=total_words)
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense

# Build the model
model = Sequential([
    Embedding(total_words, 64,
input_length=max_sequence_len-1),
    LSTM(100),
    Dense(total_words, activation='softmax')
```

```
])


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


# Train the model

history = model.fit(X, y, epochs=30, verbose=1)

def predict_next_line(seed_text, max_sequence_len, model,
tokenizer):

    token_list = tokenizer.texts_to_sequences([seed_text])[0]

    token_list = pad_sequences([token_list],
maxlen=max_sequence_len-1, padding='pre')

    predicted = model.predict(token_list, verbose=0)

    predicted_word_index = np.argmax(predicted, axis=1)[0]

    output_word = tokenizer.index_word[predicted_word_index]

    return output_word


# Get user input and predict the next line

user_input = input("Enter a line in Tamil: ")

next_line = predict_next_line(user_input, max_sequence_len,
model, tokenizer)

print("Predicted next line:", next_line)

def predict_next_line(seed_text, max_sequence_len, model,
tokenizer):

    token_list = tokenizer.texts_to_sequences([seed_text])[0]

    token_list = pad_sequences([token_list],
maxlen=max_sequence_len-1, padding='pre')

    predicted = model.predict(token_list, verbose=0)

    predicted_word_index = np.argmax(predicted, axis=1)[0]
```

```
    output_word = tokenizer.index_word[predicted_word_index]
    return output_word

# Get user input and predict the next line
user_input = input("Enter a line in Tamil: ")
next_line = predict_next_line(user_input, max_sequence_len, model, tokenizer)
print("Predicted next line:", next_line)

# Final accuracy in percentage
final_accuracy = history.history['accuracy'][-1] * 100
final_accuracy=packag.accuracy(final_accuracy)
print(f"Final Model Accuracy: {final_accuracy:.2f}%")final_accuracy=packag.accuracy(final_accuracy)
```

```
...   Final Model Accuracy: 86.57%
```