English Spell Correcting Chatbot using Natural Language Processing techniques

Sana Izumi
University of San Carlos
Email: 20101443@usc.edu.ph

***Abstract*—This project demonstrates the application of Natural Language Processing (NLP) techniques to develop a spell-correcting chatbot aimed at assisting non-native English learners in improving their vocabulary and writing skills. The chatbot employs two models: a Logistic Regression model and an LSTM-based neural network model, which were compared in terms of accuracy, precision, recall, and F1 score. The LSTM model showed significant improvements over the Logistic Regression model, achieving higher accuracy and better performance metrics. Despite these improvements, practical usage of the chatbot revealed that the suggestion accuracy needs further enhancement. Future work could focus on incorporating other neural network architectures and introducing different mechanisms to generate and compare the misspelled words dataset, to improve the overall model accuracy and effectiveness.**

I. Introduction

Natural Language Processing (NLP) is a branch of artificial intelligence (AI) utilized to analyze and interpret written and verbal human language. The primary objective is to design programs that can generate and comprehend natural language, enabling people to communicate with computers and have computers analyze their needs.[**?**] Machine learning is finding more and more applications in data analytics, such as the creation of chatbots that can mimic human user discussions by extracting relevant insights from massive databases. [**?**]

One prominent application of NLP is the development of chatbots. Chatbots utilize NLP to facilitate human-like interactions between computers and users, offering a wide range of functionalities from customer service to personal assistance. In this project, the primary function of the chatbot is spell correction. Spell correction tools are essential for enhancing written communication by automatically identifying and rectifying spelling errors. For non-native English learners, spell correction tools serve as a valuable aid, helping them to learn and use new vocabulary accurately. This chatbot will not only help users recognize and correct their spelling mistakes but also enhance their overall learning experience by reinforcing the correct spelling of words.

The primary objective of this project is to apply the concepts and techniques learned in NLP to develop a practical tool that assists non-native English learners in improving their vocabulary. This chatbot utilize various NLP concepts such as tokenization, vectorization, and neural networks to detect, correct spelling errors, and provide real-time feedback to users.

II. Relevance/Significance

A. Importance of Spell Correction for Non-Native English Learners

The ability to write correctly is fundamental to effective communication. For non-native English learners, mastering spelling is a crucial aspect of their language acquisition journey. Spell correction tools can significantly alleviate the challenges faced by these learners by providing immediate feedback and corrections. This not only helps in minimizing errors but also aids in reinforcing the correct spelling of words, thereby improving their overall proficiency in English.

Inaccurate spelling can lead to misunderstandings, reduced readability, and a lack of professionalism in written communication. In educational settings, spelling mistakes can hinder the learning process, causing confusion and impeding the acquisition of new vocabulary. Therefore, integrating effective spell correction capabilities into educational tools is vital for supporting learners in developing their language skills.

B. Enhancing Digital Communication

In the digital age, written communication is ubiquitous, spanning emails, text messages, social media posts, and more. Accurate spelling is essential for clear and professional communication. Spell correction tools embedded in chatbots can play a pivotal role in ensuring that users' written communication is free of errors. This is particularly relevant in professional and educational contexts, where precision in language is crucial.

C. Advancements in NLP and Educational Technology

The development of this spell-correcting chatbot exemplifies the practical application of advanced NLP techniques. By leveraging concepts such as tokenization, vectorization, and neural networks, this project showcases how theoretical knowledge can be transformed into a functional and beneficial tool. This not only underscores the versatility of NLP in solving real-world problems but also highlights its potential to revolutionize educational technology.

III. Methodology

A. Flowchart of Methodology

The following flowchart illustrates the steps involved in developing the spell-correcting chatbot:

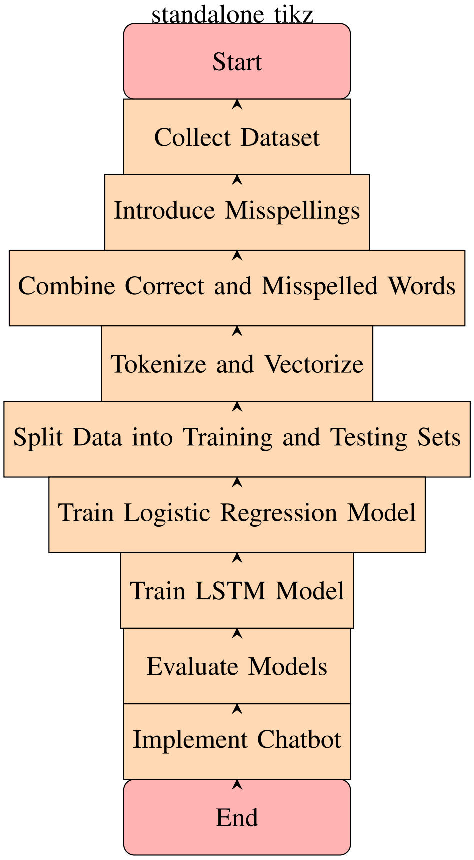

Fig. 1. Flowchart of the methodology for developing the spell-correcting chatbot.

B. Data Collection and Preprocessing

The first step in developing the spell-correcting chatbot was to collect and preprocess the data. I utilized the words corpus from the Natural Language Toolkit (NLTK), which contains a comprehensive list of English words. This dataset provided the foundation for both correctly spelled words and misspelled variants.

1) Introducing Misspellings: To simulate real-world scenarios where users may input misspelled words, I created a function to introduce random misspellings into the dataset. This function, introduce_misspellings, modifies a given word by randomly replacing one of its characters with another character from the alphabet, based on a specified error rate. The experimented error rates were 0.1(10%), 0.5(50%), and 0.8(80%). This process generated a diverse set of misspelled words to train the model effectively.

Listing 1. Data Collection and Preprocessing

```
import nltk
from nltk.corpus import words
import random

# Download the dataset if not already done
nltk.download('words')

# Load the dataset
word_list = words.words()

# Function to introduce misspellings
def introduce_misspellings(word, error_rate=0.1)
    :
    if random.random() < error_rate:
        index = random.randint(0, len(word) - 1)
        return word[:index] + random.choice('
            abcdefghijklmnopqrstuvwxyz') + word[
            index + 1:]
    return word

# Create a dataset with misspellings
misspelled_word_list = [introduce_misspellings(
    word) for word in word_list]
```

2) Combining Correct and Misspelled Words: I combined the original correctly spelled words with the newly generated misspelled words to create a comprehensive dataset. Each word in the dataset was labeled as either correct (0) or misspelled (1).

Listing 2. Data Collection and Preprocessing

```
# Combine correct and misspelled words
combined_word_list = word_list +
    misspelled_word_list
labels = [0] * len(word_list) + [1] * len(
    misspelled_word_list)
```

C. Tokenization and Vectorization

To prepare the data for training, I used tokenization and vectorization techniques. Tokenization involved breaking down the words into individual characters, which were then converted into sequences of numerical values using the Keras Tokenizer. These sequences were padded to ensure uniform length, facilitating input into the neural network.

D. Data Splitting

The dataset was split into training and testing sets, with 80% of the data used for training and 20% reserved for testing. This ensured that the model was evaluated on data it had not seen during training, providing an accurate measure of its performance.

E. Logistic Regression Model

Initially, I implemented a Logistic Regression model with a pipeline using the 'liblinear' solver. This model served as a baseline to compare the performance with the more complex LSTM-based neural network.

Listing 3. Data Collection and Preprocessing

```
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import
    train_test_split
from sklearn.linear_model import
    LogisticRegression
from sklearn.metrics import accuracy_score,
    classification_report

# Split the data
X_train, X_test, y_train, y_test =
    train_test_split(X, labels, test_size=0.2,
    random_state=42)
```

```python

    # Create a pipeline with scaling and logistic regression using 'liblinear' solver
    pipeline = Pipeline([
        ('scaler', StandardScaler(with_mean=False)),  # Scaling the data
        ('logreg', LogisticRegression(max_iter=1000, solver='liblinear'))  # Use 'liblinear' solver
    ])

    # Train the model
    pipeline.fit(X_train, y_train)

    # Evaluate the model
    y_pred = pipeline.predict(X_test)
    print(f"Accuracy: {accuracy_score(y_test, y_pred)}")
    print(classification_report(y_test, y_pred))
```

F. LSTM-Based Neural Network Model

To optimize and improve the accuracy, I built an LSTM-based neural network model according to Dhankhar(2018)[?]. The model architecture consisted of an embedding layer, two LSTM layers, and a dense output layer with a sigmoid activation function.

¥beginitemize ¥item Embedding Layer: Converts the input sequences into dense vectors of fixed size. ¥item LSTM Layers: Capture the sequential dependencies in the character sequences. ¥item Dense Output Layer: Provides a binary classification output (correct or misspelled). ¥enditemize

Listing 4. Data Collection and Preprocessing

```python
    from keras.models import Sequential
    from keras.layers import Embedding, LSTM, Dense

    # Build LSTM model
    model = Sequential()
    model.add(Embedding(input_dim=len(tokenizer.word_index) + 1, output_dim=50, input_length=max_seq_length))
    model.add(LSTM(100, return_sequences=True))
    model.add(LSTM(100))
    model.add(Dense(1, activation='sigmoid'))

    # Compile the model
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

G. Training and Evaluation

I experimented and turned with different hyperparameters, such as the error rate for generating misspellings, the number of epochs, and the batch size, to find the optimal configuration for the LSTM model. The final model was trained for 10 epochs with a batch size of 64 using error rate 0.8(80%). The validation dataset was used to monitor the model's performance during training. After training, the model's accuracy was evaluated on the test dataset.

Listing 5. Data Collection and Preprocessing

```python
    # Train the model
    history = model.fit(X_train, y_train, epochs=10, batch_size=64, validation_data=(X_test, y_test))

    # Evaluate the model
    loss, accuracy = model.evaluate(X_test, y_test)
    print(f"LSTM Model Accuracy: {accuracy}")

    # Predict and evaluate
    y_pred = (model.predict(X_test) > 0.5).astype("int32")
    print(f"LSTM Model Accuracy: {accuracy_score(y_test, y_pred)}")
    print(classification_report(y_test, y_pred))
```

H. Chatbot Implementation

The chatbot function was designed to take user input, preprocess it, and use the trained model to predict whether the input word was correctly spelled or not. If the word is misspelled, the chatbot suggest the most similar correctly spelled word based on character similarity and also displayed its matching accuracy. If the word is spelled correctly, the chatbot return "Nothing Wrong!".

Listing 6. Data Collection and Preprocessing

```python
    # Chatbot function to correct spelling
    def correct_spelling(input_word):
        input_seq = tokenizer.texts_to_sequences([input_word])
        input_seq = pad_sequences(input_seq, maxlen=max_seq_length)

        prediction = model.predict(input_seq)

        if prediction < 0.5:
            return "Nothing Wrong!"

        # Find the most similar word
        similarities = []
        for word in word_list:
            if len(word) == len(input_word):
                sim = sum(1 for a, b in zip(word, input_word) if a == b)
                similarities.append((word, sim))
        if not similarities:
            return "No suggestions available."

        best_match = max(similarities, key=lambda x: x[1])[0]
        accuracy = max(similarities, key=lambda x: x[1])[1] / len(input_word) * 100

        return f"Suggested Correction: {best_match}, Matching Accuracy: {accuracy:.2f}%"

    # Simple chatbot interface
    def chatbot():
        print("Welcome to the English Correcting Chatbot! Type 'exit' to quit.")
        while True:
            user_input = input("Enter a word: ").strip()
            if user_input.lower() == 'exit':
                break
            correction = correct_spelling(user_input)
            print(correction)

    # Run the chatbot
    chatbot()
```

IV. Results and Analysis

This section provides a comprehensive overview of the evaluation criteria, results, and analysis for both the Logistic

Regression and LSTM-based neural network models, along with observations from the chatbot usage. By detailing the different conditions and their corresponding metrics, the analysis highlights the improvements achieved with the LSTM model and provides insights into the model's performance under various configurations.

A. Evaluation Criteria

Based on the study by Katsumata(2020)[**?**], I used these metrics to evaluate the performance of the spell-correcting chatbot:

- **Accuracy**: The proportion of correctly identified instances (correct or misspelled) out of the total instances.
- **Precision**: The ratio of correctly predicted positive observations to the total predicted positives.
- **Recall**: The ratio of correctly predicted positive observations to all observations in the actual class.
- **F1 Score**: The weighted average of Precision and Recall, providing a balance between the two.

These metrics provide a comprehensive evaluation of the models' performance, considering both their ability to correctly identify misspellings and their reliability in predicting the correct class.

B. Logistic Regression Model

I experimented with the Logistic Regression model using three different error rates for introducing misspellings: 10%, 50%, and 80%. Regardless of the error rate, the results were consistent across all trials.

TABLE I
EVALUATION METRICS

Metric	Value
Accuracy	0.4645
Precision (Class 0)	0.47
Precision (Class 1)	0.44
Recall (Class 0)	0.65
Recall (Class 1)	0.28
F1-score (Class 0)	0.55
F1-score (Class 1)	0.35
Support (Class 0)	47305
Support (Class 1)	47390

```
Accuracy: 0.4645123818575426
              precision    recall  f1-score   support

           0       0.47      0.65      0.55     47305
           1       0.44      0.28      0.35     47390

    accuracy                           0.46     94695
   macro avg       0.46      0.46      0.45     94695
weighted avg       0.46      0.46      0.45     94695
```

Fig. 2. Evaluation results of Logistic Regression model.

1) Logistic Regression Model Results:

C. LSTM-Based Neural Network Model

To optimize and improve accuracy, I built an LSTM-based neural network model. I experimented with various combinations of error rates, epochs, and batch sizes:

- **(error rate = 0.1, epochs = 10, batch size = 64)**
- **(error rate = 0.5, epochs = 10, batch size = 64)**
- **(error rate = 0.5, epochs = 10, batch size = 32)**
- **(error rate = 0.8, epochs = 10, batch size = 64)**

1) LSTM Model Results: 1. **(error rate = 0.1, epochs = 10, batch size = 64)**

TABLE II
EVALUATION METRICS

Metric	Value
Accuracy	0.5153
Precision (Class 0)	0.51
Precision (Class 1)	0.53
Recall (Class 0)	0.76
Recall (Class 1)	0.27
F1-score (Class 0)	0.61
F1-score (Class 1)	0.36
Support (Class 0)	47305
Support (Class 1)	47390

```
[nltk_data] Downloading package words to /root/nltk_data...
[nltk_data]   Package words is already up-to-date!
Epoch 1/10
5919/5919 [==============================] - 529s 89ms/step - loss: 0.6924 - accuracy: 0.5068 - val_loss: 0.6897 - val_accuracy: 0.5101
Epoch 2/10
5919/5919 [==============================] - 527s 89ms/step - loss: 0.6876 - accuracy: 0.5116 - val_loss: 0.6864 - val_accuracy: 0.5136
Epoch 3/10
5919/5919 [==============================] - 529s 89ms/step - loss: 0.6861 - accuracy: 0.5139 - val_loss: 0.6857 - val_accuracy: 0.5155
Epoch 4/10
5919/5919 [==============================] - 532s 90ms/step - loss: 0.6856 - accuracy: 0.5145 - val_loss: 0.6853 - val_accuracy: 0.5157
Epoch 5/10
5919/5919 [==============================] - 528s 89ms/step - loss: 0.6850 - accuracy: 0.5157 - val_loss: 0.6850 - val_accuracy: 0.5161
Epoch 6/10
5919/5919 [==============================] - 529s 89ms/step - loss: 0.6846 - accuracy: 0.5165 - val_loss: 0.6848 - val_accuracy: 0.5161
Epoch 7/10
5919/5919 [==============================] - 527s 89ms/step - loss: 0.6842 - accuracy: 0.5169 - val_loss: 0.6847 - val_accuracy: 0.5163
Epoch 8/10
5919/5919 [==============================] - 526s 89ms/step - loss: 0.6838 - accuracy: 0.5181 - val_loss: 0.6846 - val_accuracy: 0.5170
Epoch 9/10
5919/5919 [==============================] - 522s 88ms/step - loss: 0.6835 - accuracy: 0.5189 - val_loss: 0.6847 - val_accuracy: 0.5170
Epoch 10/10
5919/5919 [==============================] - 567s 96ms/step - loss: 0.6832 - accuracy: 0.5195 - val_loss: 0.6846 - val_accuracy: 0.5153
2960/2960 [==============================] - 45s 15ms/step - loss: 0.6846 - accuracy: 0.5153
Accuracy: 0.5152648091316223
2960/2960 [==============================] - 46s 15ms/step
Accuracy: 0.5152647975077882
              precision    recall  f1-score   support

           0       0.51      0.76      0.61     47305
           1       0.53      0.27      0.36     47390

    accuracy                           0.52     94695
   macro avg       0.52      0.52      0.48     94695
weighted avg       0.52      0.52      0.48     94695
```

Fig. 3. Evaluation results of LSTM model with error rate 0.1, epochs 10, batch size 64.

2. **(error rate = 0.5, epochs = 10, batch size = 64)**

TABLE III
EVALUATION METRICS

Metric	Value
Accuracy	0.6380
Precision (Class 0)	0.59
Precision (Class 1)	0.80
Recall (Class 0)	0.91
Recall (Class 1)	0.37
F1-score (Class 0)	0.71
F1-score (Class 1)	0.50
Support (Class 0)	47305
Support (Class 1)	47390

3. **(error rate = 0.5, epochs = 10, batch size = 32)**
4. **(error rate = 0.8, epochs = 10, batch size = 64)**

```
Epoch 1/10
5919/5919 [==============================] - 567s 95ms/step - loss: 0.6515 - accuracy: 0.5810 - val_loss: 0.6352 - val_accuracy: 0.5997
Epoch 2/10
5919/5919 [==============================] - 582s 98ms/step - loss: 0.6301 - accuracy: 0.6087 - val_loss: 0.6259 - val_accuracy: 0.6139
Epoch 3/10
5919/5919 [==============================] - 578s 98ms/step - loss: 0.6198 - accuracy: 0.6205 - val_loss: 0.6174 - val_accuracy: 0.6215
Epoch 4/10
5919/5919 [==============================] - 536s 91ms/step - loss: 0.6126 - accuracy: 0.6282 - val_loss: 0.6137 - val_accuracy: 0.6255
Epoch 5/10
5919/5919 [==============================] - 574s 97ms/step - loss: 0.6066 - accuracy: 0.6342 - val_loss: 0.6093 - val_accuracy: 0.6322
Epoch 6/10
5919/5919 [==============================] - 580s 98ms/step - loss: 0.6009 - accuracy: 0.6397 - val_loss: 0.6064 - val_accuracy: 0.6350
Epoch 7/10
5919/5919 [==============================] - 550s 93ms/step - loss: 0.5959 - accuracy: 0.6448 - val_loss: 0.6050 - val_accuracy: 0.6372
Epoch 8/10
5919/5919 [==============================] - 572s 97ms/step - loss: 0.5911 - accuracy: 0.6493 - val_loss: 0.6045 - val_accuracy: 0.6383
Epoch 9/10
5919/5919 [==============================] - 573s 97ms/step - loss: 0.5867 - accuracy: 0.6532 - val_loss: 0.6038 - val_accuracy: 0.6394
Epoch 10/10
5919/5919 [==============================] - 531s 90ms/step - loss: 0.5831 - accuracy: 0.6562 - val_loss: 0.6035 - val_accuracy: 0.6392
```

```
2960/2960 [==============================] - 47s 16ms/step - loss: 0.6095 - accuracy: 0.6380
Accuracy: 0.6379956603050232
2960/2960 [==============================] - 46s 15ms/step
Accuracy: 0.6379956703099424
              precision    recall  f1-score   support

           0       0.59      0.91      0.71     47305
           1       0.80      0.37      0.50     47390

    accuracy                           0.64     94695
   macro avg       0.69      0.64      0.61     94695
weighted avg       0.69      0.64      0.61     94695
```

Fig. 4. Evaluation results of LSTM model with error rate 0.5, epochs 10, batch size 64.

TABLE IV
EVALUATION METRICS

Metric	Value
Accuracy	0.6408
Precision (Class 0)	0.59
Precision (Class 1)	0.81
Recall (Class 0)	0.91
Recall (Class 1)	0.37
F1-score (Class 0)	0.72
F1-score (Class 1)	0.51
Support (Class 0)	47305
Support (Class 1)	47390

```
Epoch 1/10
11837/11837 [==============================] - 617s 52ms/step - loss: 0.6477 - accuracy: 0.5860 - val_loss: 0.6319 - val_accuracy: 0.6070
Epoch 2/10
11837/11837 [==============================] - 608s 51ms/step - loss: 0.6259 - accuracy: 0.6140 - val_loss: 0.6206 - val_accuracy: 0.6193
Epoch 3/10
11837/11837 [==============================] - 566s 48ms/step - loss: 0.6158 - accuracy: 0.6253 - val_loss: 0.6156 - val_accuracy: 0.6259
Epoch 4/10
11837/11837 [==============================] - 607s 51ms/step - loss: 0.6075 - accuracy: 0.6348 - val_loss: 0.6096 - val_accuracy: 0.6337
Epoch 5/10
11837/11837 [==============================] - 605s 51ms/step - loss: 0.6008 - accuracy: 0.6414 - val_loss: 0.6061 - val_accuracy: 0.6359
Epoch 6/10
11837/11837 [==============================] - 563s 48ms/step - loss: 0.5954 - accuracy: 0.6462 - val_loss: 0.6044 - val_accuracy: 0.6391
Epoch 7/10
11837/11837 [==============================] - 604s 51ms/step - loss: 0.5906 - accuracy: 0.6504 - val_loss: 0.6142 - val_accuracy: 0.6365
Epoch 8/10
11837/11837 [==============================] - 606s 51ms/step - loss: 0.5869 - accuracy: 0.6538 - val_loss: 0.6048 - val_accuracy: 0.6405
Epoch 9/10
11837/11837 [==============================] - 572s 48ms/step - loss: 0.5833 - accuracy: 0.6564 - val_loss: 0.6039 - val_accuracy: 0.6399
Epoch 10/10
11837/11837 [==============================] - 615s 52ms/step - loss: 0.5802 - accuracy: 0.6592 - val_loss: 0.6047 - val_accuracy: 0.6408
2960/2960 [==============================] - 48s 16ms/step - loss: 0.6047 - accuracy: 0.6408
Accuracy: 0.6407941579818726
2960/2960 [==============================] - 47s 16ms/step
Accuracy: 0.6407941285178732
              precision    recall  f1-score   support

           0       0.59      0.91      0.72     47305
           1       0.81      0.37      0.51     47390

    accuracy                           0.64     94695
   macro avg       0.70      0.64      0.61     94695
weighted avg       0.70      0.64      0.61     94695
```

Fig. 5. Evaluation results of LSTM model with error rate 0.5, epochs 10, batch size 32.

D. Analysis

1) Logistic Regression Model: The Logistic Regression model achieved an accuracy of approximately 46.45%, with a precision of 0.47 for class 0 (correct words) and 0.44 for class 1 (misspelled words). The recall for class 0 was higher (0.65) compared to class 1 (0.28), indicating that the model was better

TABLE V
EVALUATION METRICS

Metric	Class 0	Class 1	Macro Avg	Weighted Avg
Accuracy	0.7451			
Precision	0.69	0.84	0.77	0.77
Recall	0.89	0.60	0.75	0.75
F1-score	0.78	0.70	0.74	0.74
Support	47305	47390	94695	94695

```
Epoch 1/10
5919/5919 [==============================] - 601s 101ms/step - loss: 0.6033 - accuracy: 0.6455 - val_loss: 0.5794 - val_accuracy: 0.6721
Epoch 2/10
5919/5919 [==============================] - 592s 100ms/step - loss: 0.5610 - accuracy: 0.6909 - val_loss: 0.5489 - val_accuracy: 0.7020
Epoch 3/10
5919/5919 [==============================] - 553s 93ms/step - loss: 0.5378 - accuracy: 0.7131 - val_loss: 0.5321 - val_accuracy: 0.7181
Epoch 4/10
5919/5919 [==============================] - 602s 102ms/step - loss: 0.5188 - accuracy: 0.7296 - val_loss: 0.5196 - val_accuracy: 0.7285
Epoch 5/10
5919/5919 [==============================] - 586s 99ms/step - loss: 0.5042 - accuracy: 0.7415 - val_loss: 0.5122 - val_accuracy: 0.7360
Epoch 6/10
5919/5919 [==============================] - 584s 99ms/step - loss: 0.4927 - accuracy: 0.7497 - val_loss: 0.5070 - val_accuracy: 0.7400
Epoch 7/10
5919/5919 [==============================] - 588s 99ms/step - loss: 0.4822 - accuracy: 0.7576 - val_loss: 0.5049 - val_accuracy: 0.7458
Epoch 8/10
5919/5919 [==============================] - 583s 99ms/step - loss: 0.4736 - accuracy: 0.7640 - val_loss: 0.5108 - val_accuracy: 0.7409
Epoch 9/10
5919/5919 [==============================] - 589s 99ms/step - loss: 0.4659 - accuracy: 0.7692 - val_loss: 0.5040 - val_accuracy: 0.7460
Epoch 10/10
5919/5919 [==============================] - 593s 100ms/step - loss: 0.4583 - accuracy: 0.7746 - val_loss: 0.5098 - val_accuracy: 0.7451
2960/2960 [==============================] - 49s 16ms/step - loss: 0.5098 - accuracy: 0.7451
Accuracy: 0.7451396584510803
2960/2960 [==============================] - 48s 16ms/step
Accuracy: 0.7451396589049052
              precision    recall  f1-score   support

           0       0.69      0.89      0.78     47305
           1       0.84      0.60      0.70     47390

    accuracy                           0.75     94695
   macro avg       0.77      0.75      0.74     94695
weighted avg       0.77      0.75      0.74     94695
```

Fig. 6. Evaluation results of LSTM model with error rate 0.8, epochs 10, batch size 64.

at identifying correctly spelled words than misspelled ones. The F1 score, a balance between precision and recall, was moderate for class 0 (0.55) and lower for class 1 (0.35).

Overall, the Logistic Regression model provided a baseline performance but lacked the capability to accurately detect and correct misspelled words, regardless of the error rate used.

2) LSTM-Based Neural Network Model: The LSTM model showed significant improvement over the Logistic Regression model, with varying results based on the error rate and batch size:

1. **(error rate = 0.1, epochs = 10, batch size = 64)**: Accuracy was 51.53%, with better performance for class 0 (precision: 0.51, recall: 0.76) compared to class 1 (precision: 0.53, recall: 0.27).

2. **(error rate = 0.5, epochs = 10, batch size = 64)**: Accuracy increased to 63.80%, with notable improvements in class 1 detection (precision: 0.80, recall: 0.37).

3. **(error rate = 0.5, epochs = 10, batch size = 32)**: Slightly better accuracy at 64.08%, with similar performance metrics as the previous configuration.

4. **(error rate = 0.8, epochs = 10, batch size = 64)**: The highest accuracy achieved was 74.51%, with significant improvements in both precision and recall for class 1 (precision: 0.84, recall: 0.60).

These results indicate that the LSTM model is more effective at handling higher error rates, showing robust performance in detecting and correcting misspelled words. The choice of batch size also played a role, with smaller batch sizes slightly improving accuracy and F1 scores.

E. Observations from Chatbot Usage

Despite the improved metrics with the LSTM model, the real-world usage of the chatbot revealed that the suggestion accuracy was quite low. This discrepancy highlights a common challenge in NLP and machine learning: metrics may not fully capture the practical effectiveness of a model. User experience can often expose limitations that quantitative evaluations might overlook.

```
Welcome to the English Correcting Chatbot! Type 'exit' to quit.
Enter a word: colect
1/1 [==============================] - 0s 31ms/step
Nothing Wrong!
Enter a word: hallo
1/1 [==============================] - 0s 31ms/step
Nothing Wrong!
Enter a word: heplo
1/1 [==============================] - 0s 30ms/step
Nothing Wrong!
Enter a word: wordd
1/1 [==============================] - 0s 43ms/step
Nothing Wrong!
Enter a word: wolrd
1/1 [==============================] - 0s 31ms/step
Suggested Correction: board, Matching Accuracy: 60.00%
Enter a word: vread
1/1 [==============================] - 0s 26ms/step
Suggested Correction: aread, Matching Accuracy: 80.00%
Enter a word: pythgn
1/1 [==============================] - 0s 28ms/step
Suggested Correction: python, Matching Accuracy: 83.33%
Enter a word: misspellll
1/1 [==============================] - 0s 28ms/step
Suggested Correction: misspender, Matching Accuracy: 60.00%
Enter a word: eixt
1/1 [==============================] - 0s 31ms/step
Suggested Correction: aint, Matching Accuracy: 50.00%
Enter a word:
```

Fig. 7. Example Chatbot Usage.

V. CONCLUSION

This project demonstrated the application of Natural Language Processing (NLP) techniques to develop a spell-correcting chatbot aimed at assisting non-native English learners in improving their vocabulary and writing skills. The primary focus was on implementing and comparing two models: a Logistic Regression model and an LSTM-based neural network model. While the LSTM model showed significant improvements over the Logistic Regression model in terms of accuracy, precision, recall, and F1 score, practical usage of the chatbot indicated that suggestion accuracy needs further improvement. Future work could focus on incorporating ither different neural network architectures and introducing different mechanisms to generate and compare the misspelled words dataset, to enhance the model accuracy.