Subtheme Sentiment Analysis Assignment Documentation

By- Dev Kumar Maan

Assignment Overview

The provided document outlines the steps taken to develop a subtheme sentiment analysis model using deep learning techniques. The project's primary goal is to predict the sentiment (positive or negative) of textual reviews.

Structure and Organization

The document is well-organized with clear sections: Introduction, Data Collection, Data Pre-processing, Model Development, Model Evaluation, Deployment, and Conclusion

1. Introduction

Sentiment analysis, also known as opinion mining, is a natural language processing (NLP) task that involves identifying and classifying subjective information in text data. It is widely used in various applications such as customer feedback analysis, social media monitoring, and market research. This project aims to develop a sentiment analysis model to predict whether a textual review expresses a positive or negative sentiment. The model is built using deep learning techniques and implemented with TensorFlow and Keras.

2. Data Collection

- The dataset is available as name ‘Evaluation-dataset.csv’, which was provided in at with the assignment.
- The dataset is loaded in Notebook using pandas methods as follows

DataSet Read

In [2]:
```
# Create a list of column names for a DataFrame
# The list will contain column names as 'Col0', 'Col1', ..., 'Col14' beacue our dataset does not have any column names
columns_for_dataframe = ['Col' + str(index) for index in range(15)]

# Insert 'Review' at the beginning of the column list
columns_for_dataframe.insert(0, 'Review')
```

In [3]:
```
# Read the dataset from the CSV file and assign column names from 'columns_for_dataframe'
dataframe_evaluation = pd.read_csv('Evaluation-dataset.csv', names=columns_for_dataframe)

# Display the first few rows of the DataFrame to verify the content
dataframe_evaluation.head()
```

Out[3]:

	Review	Col0	Col1	Col2	Col3	Col4	Col5	Col6	Col7	Col8	Col9	Col10	Col11	Col12	Col13	Col14
0	Tires where delivered to the garage of my choi...	garage service positive	ease of booking positive	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	Easy Tyre Selection Process, Competitive Prici...	garage service positive	value for money positive	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	Very easy to use and good value for money.	value for money positive	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	Really easy and convenient to arrange	ease of booking positive	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	It was so easy to select tyre sizes and arrang...	location positive	value for money positive	ease of booking positive	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

3. Data Pre-processing

- **Tokenization:** the process of breaking down text into individual units such as words or subwords, which are then used as input for the model to analyze sentiment. This step is crucial for transforming raw text into a format that can be effectively processed by machine learning algorithms.

- **Padding:** Ensures that all input sequences have the same length by adding special tokens to shorter sequences. This standardization allows the model to process batches of data efficiently, regardless of the original sequence lengths
- **Encoding**: Involves converting tokens into numerical representations that can be processed by machine learning models. This step typically uses techniques such as word embeddings or one-hot encoding to transform textual data into vectors.
- **Splitting**: Divide the dataset into training and testing sets.

In [18]:
```
# Filter out rows containing positive sentiments
positive = review_distribution[review_distribution['Reviews'].str.contains("positive")]

# Filter out rows containing negative sentiments
negative = review_distribution[review_distribution['Reviews'].str.contains("negative")]

# Calculate the total number of sentiments with positive polarity tag
total_positive_sentiments = positive['Sum'].sum()

# Calculate the total number of sentiments with negative polarity tag
total_negative_sentiments = negative['Sum'].sum()

# Print the total number of sentiments with positive polarity tag
print('Total number of sentiments with positive polarity tag: {}'.format(total_positive_sentiments))

# Print the total number of sentiments with negative polarity tag
print('Total number of sentiments with negative polarity tag: {}'.format(total_negative_sentiments))

# Print the total size of the data
print('Total size of data: {}'.format(dataframe_evaluation.shape[0]))

# Here we can see the class imbalance, as both negative and positive sentiments are present.
```

```
Total number of sentiments with positive polarity tag: 11889.0
Total number of sentiments with negative polarity tag: 2611.0
Total size of data: 10132
```

```
# Tokenization and Padding

# Initialize a Tokenizer object
tokenizer = Tokenizer()

# Fit the tokenizer on the text data in the 'Review' column of the DataFrame
tokenizer.fit_on_texts(dataframe_evaluation['Review'])

# Convert text sequences to token sequences
token_sequences = tokenizer.texts_to_sequences(dataframe_evaluation['Review'])

# Determine the maximum sequence length
max_sequence_length = max([len(sequence) for sequence in token_sequences])

# Pad the sequences to have uniform length for model input
padded_sequences = pad_sequences(token_sequences, maxlen=max_sequence_length, padding='post')
```

```
# Split the data into training and testing sets

# Assign features and target variable to training and testing sets
X_train, X_test, y_train, y_test = train_test_split(padded_sequences, dataframe_evaluation['Contains_Redacted'], test_size=0.25)
```

4. Model Defining

- **Motivation:** Recurrent Neural Networks (RNNs), particularly Long Short-Term Memory (LSTM) networks, are effective for sequence data like text. The model architecture is designed to capture the temporal dependencies in the data and perform binary classification.

Approach:

- **Embedding Layer**: Converts integer representations of words into dense vectors of fixed size.
- **LSTM Layer:** Captures long-term dependencies in the text.
- **Batch Normalization**: Normalizes activations to improve training stability and performance.
- **Dropout**: Prevents overfitting by randomly dropping units during training.
- **Dense Layers**: Further layers for nonlinear transformations and classification.

```
# Define the RNN model
model = Sequential()
model.add(Embedding(input_dim=len(tokenizer.word_index)+1, output_dim=100))
model.add(LSTM(64, dropout=0.2, recurrent_dropout=0.2))
model.add(BatchNormalization())
model.add(Dropout(0.1))
model.add(Dense(32, activation='relu', kernel_regularizer='l2'))
model.add(Dense(1, activation='sigmoid'))  # Output layer with binary classifi
```

5. Model Compiling

- Compiled themodel using RMSprop optimizer in order to increase accuracy.
- Binary Cross entropy Loss function has been used in order for better metrics analysis
- Accuracy was used as the metrics

Model Compiling

In [72]:
```
model.compile(optimizer=RMSprop(learning_rate=0.001),
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

6. Model Training

- Model was trained at first having set the batch size as 64 and number of epochs as 10.

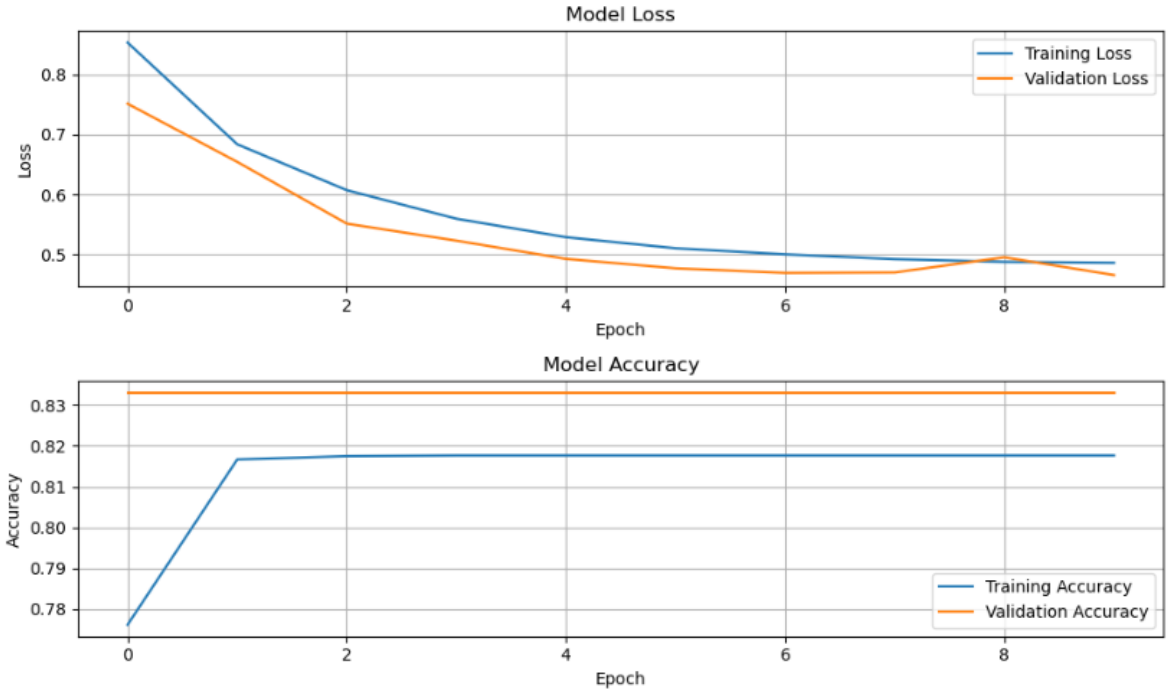
Model Training

```
# Fit the model to the training data, using validation data for evaluation
training_history = model.fit(
    X_train,  # Training data
    y_train,  # Training labels
    epochs=10,  # Number of epochs to train
    batch_size=64,  # Size of batches during training
    validation_data=(X_test, y_test),  # Validation data
    callbacks=[
        EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True),  # Stop training if validation loss doesn't imp
        ModelCheckpoint('best_model.keras', monitor='val_loss', save_best_only=True)  # Save the best model based on validation l
    ]
)
```

```
Epoch 1/10
96/96 ━━━━━━━━━━━━━━━━━━━━ 30s 289ms/step - accuracy: 0.6926 - loss: 0.9560 - val_accuracy: 0.8332 - val_loss: 0.7511
Epoch 2/10
96/96 ━━━━━━━━━━━━━━━━━━━━ 30s 316ms/step - accuracy: 0.8099 - loss: 0.7174 - val_accuracy: 0.8332 - val_loss: 0.6543
Epoch 3/10
96/96 ━━━━━━━━━━━━━━━━━━━━ 35s 363ms/step - accuracy: 0.8121 - loss: 0.6309 - val_accuracy: 0.8332 - val_loss: 0.5515
Epoch 4/10
96/96 ━━━━━━━━━━━━━━━━━━━━ 29s 306ms/step - accuracy: 0.8201 - loss: 0.5690 - val_accuracy: 0.8332 - val_loss: 0.5232
Epoch 5/10
96/96 ━━━━━━━━━━━━━━━━━━━━ 30s 317ms/step - accuracy: 0.8310 - loss: 0.5155 - val_accuracy: 0.8332 - val_loss: 0.4929
Epoch 6/10
96/96 ━━━━━━━━━━━━━━━━━━━━ 31s 327ms/step - accuracy: 0.8123 - loss: 0.5205 - val_accuracy: 0.8332 - val_loss: 0.4772
Epoch 7/10
96/96 ━━━━━━━━━━━━━━━━━━━━ 32s 329ms/step - accuracy: 0.8171 - loss: 0.5039 - val_accuracy: 0.8332 - val_loss: 0.4695
Epoch 8/10
96/96 ━━━━━━━━━━━━━━━━━━━━ 30s 308ms/step - accuracy: 0.8166 - loss: 0.4945 - val_accuracy: 0.8332 - val_loss: 0.4703
Epoch 9/10
```

7. Model Evaluation

- Evaluate the trained model on the testing dataset.
- The Accuracy came out to be 83.78 % with Loss = 0.4601



8. Model Training (By changing Parameters)

- Model was trained at first having set the batch size as 32 and number of epochs as 25

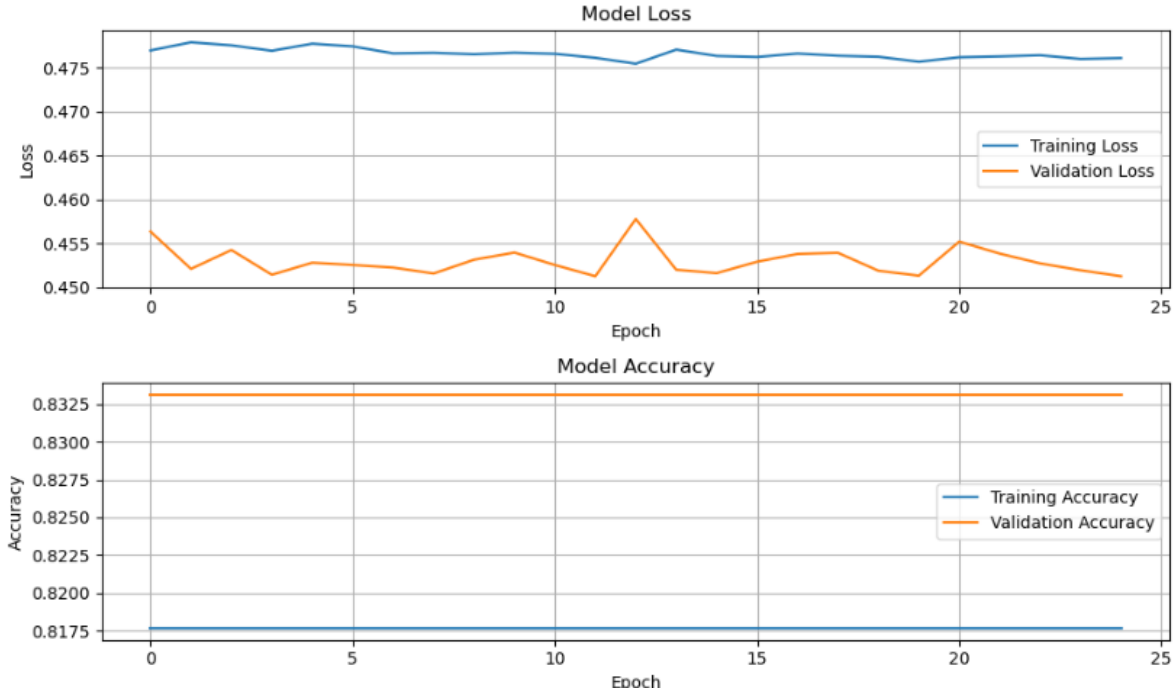
Model Training using differnt parameters

```
# Fit the model to the training data, using validation data for evaluation
training_history_2 = model.fit(
    X_train,  # Training data
    y_train,  # Training labels
    epochs=25,  # Number of epochs to train
    batch_size=32,  # Size of batches during training
    validation_data=(X_test, y_test),  # Validation data
    callbacks=[
        ModelCheckpoint('best_model_2.keras', monitor='val_loss', save_best_only=True)  # Save the best model based on validatior
    ]
)
```

```
Epoch 1/25
191/191 ━━━━━━━━━━━━━━━━━━━━ 32s 168ms/step - accuracy: 0.8141 - loss: 0.4820 - val_accuracy: 0.8332 - val_loss: 0.4564
Epoch 2/25
191/191 ━━━━━━━━━━━━━━━━━━━━ 34s 178ms/step - accuracy: 0.8170 - loss: 0.4780 - val_accuracy: 0.8332 - val_loss: 0.4521
Epoch 3/25
191/191 ━━━━━━━━━━━━━━━━━━━━ 36s 187ms/step - accuracy: 0.8216 - loss: 0.4726 - val_accuracy: 0.8332 - val_loss: 0.4543
Epoch 4/25
191/191 ━━━━━━━━━━━━━━━━━━━━ 34s 179ms/step - accuracy: 0.8204 - loss: 0.4722 - val_accuracy: 0.8332 - val_loss: 0.4515
Epoch 5/25
191/191 ━━━━━━━━━━━━━━━━━━━━ 37s 193ms/step - accuracy: 0.8123 - loss: 0.4857 - val_accuracy: 0.8332 - val_loss: 0.4528
Epoch 6/25
191/191 ━━━━━━━━━━━━━━━━━━━━ 37s 195ms/step - accuracy: 0.8201 - loss: 0.4733 - val_accuracy: 0.8332 - val_loss: 0.4526
Epoch 7/25
191/191 ━━━━━━━━━━━━━━━━━━━━ 36s 186ms/step - accuracy: 0.8130 - loss: 0.4836 - val_accuracy: 0.8332 - val_loss: 0.4523
Epoch 8/25
191/191 ━━━━━━━━━━━━━━━━━━━━ 35s 181ms/step - accuracy: 0.8196 - loss: 0.4740 - val_accuracy: 0.8332 - val_loss: 0.4516
Epoch 9/25
```

9. Model Evaluation (By changing Parameters)

- Evaluate the trained model on the testing dataset.
- The Accuracy came out to be 83.31 % with Loss = 0.4441, thus we can observe that our loss decreases by changing parameters.



10. Model Shortcomings

1. **Limited Feature Representation**:
    - The current feature engineering might not be sufficient to capture the complexity of the data, leading to suboptimal model performance.
2. **Hyperparameters**:
    - Default hyperparameters might not be well-suited for the given dataset, leading to suboptimal performance.
3. **Data Imbalance**:
    - The dataset is imbalanced, the model might be biased towards the majority class, affecting the accuracy and overall performance as the ratio of positive to negative subtheme is 4.4 which leads to less perfect model.

11. Proposed Improvements

1. **Improve Feature Engineering**:
    - Use more sophisticated feature extraction techniques like TF-IDF or word embeddings to better represent the textual data.
2. **Hyperparameter Tuning**:
    - Use techniques like grid search or random search to find the best hyperparameters for the model.
3. **Handle Class Imbalance**:
    - Use techniques like oversampling, undersampling, or class weighting to address class imbalance.
4. **Cross-Validation**:
    - Implement cross-validation to ensure that the model's performance is consistent across different subsets of the data.
5. **Advanced Models**:
    - Consider using more advanced models such as ensemble methods (Random Forest, Gradient Boosting) or deep learning models (LSTM, BERT for text data).
6. **Learning Rate Schedulers**:
    - Use learning rate schedulers to adjust the learning rate during training for better convergence.

12. Conclusion

The overall approach ensures that the text data is clean and uniformly formatted before being fed into a machine learning model. The tokenization and padding steps convert text into a suitable numerical format for the RNN. The chosen model architecture (LSTM) is well-suited for handling sequential data and is designed to prevent overfitting and enhance training stability. This methodical process aims to achieve robust and accurate predictions for the binary classification task at hand.