A Tutorial on Natural Language Processing (NLP) – Sentiment Analysis Using LSTM

By: Bhavin Thakur (Student ID: 23079699)

GitHub Link: https://github.com/bhavinthakur29/nlp-tutorial-mlnn

Let's start with the introduction,

What is Natural Language Processing (NLP)?

NLP or natural language processing basically refers to working with machines to automatically process and make sense of human language. In other words, it is the procedure that changes text to structured data so that machine learning models draw inferences of kinds that are human-like.

Example:

- When you type "What's the weather today?" into google, it'll decipher from NLP that you're inquiring about weather, then extract entities like 'weather,' 'today,' etc.
- It just helps Siri comprehend the request of the user when he says, "Play some music." Here the action is that Siri knows what to do after capturing the intent: that is, playing music.

NLP is widely used in:

- **Spam detection:** Identify whether an email is spam or not based on the text content.
- **Machine translation:** Translation of text from one language to another (e.g., English to French).
- **Speech recognition:** Transformation of spoken words into texts.
- **Text summarization:** Reduction of a large text document into a short summary.

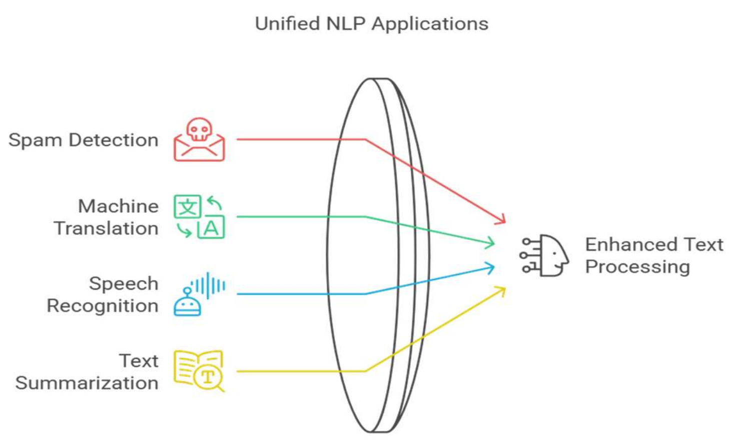

Fig 1. Applications of NLP

What is Sentiment Analysis?

The ultimate outcome of sentiment analysis is to define the emotional tone behind any text. In fact, most of the time, it is used with regard to some customer feedbacks, social media trends, and product reviews.

Example:

1. *"The movie was absolutely amazing!"* → Positive Sentiment
2. *"The product stopped working after two days. Terrible experience."* → Negative Sentiment
3. *"It's a phone. It works."* → Neutral Sentiment

One example of how a company would apply sentiment analysis is in monitoring how the clients view its products in Amazon. Assuming the reviews on many counts for a certain product are positive, it shows that the product is accepted by the customers. On the contrary, reviews that keep giving a consistent signal of negativity show that there could be a problem with that product.

Sentiment analysis models classify text using techniques such as:

- **Rule-based approaches-** Counting positive and negative words
- **Machine learning models-** Training model with positive and negative labeled data to classify any new text
- **Deep learning models-** Complex model like LSTM and transformers work to capture context and tone such that they could predict the outcome of the customer as per the emotion expression.

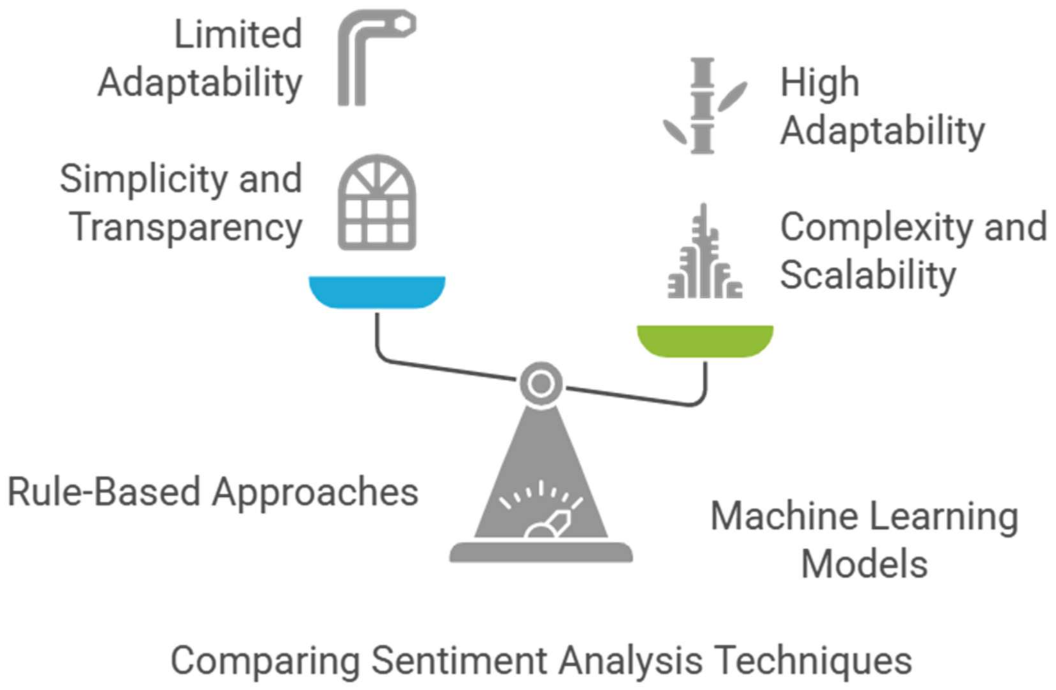
Limited Adaptability
Simplicity and Transparency
High Adaptability
Complexity and Scalability
Rule-Based Approaches
Machine Learning Models
Comparing Sentiment Analysis Techniques

Fig 2. Comparing Sentimental Analysis Techniques

2. Background

2.1 Natural Language Processing Pipeline

The NLP pipeline consists of several phases that transform raw text into structured data which id suitable for machine learning models.

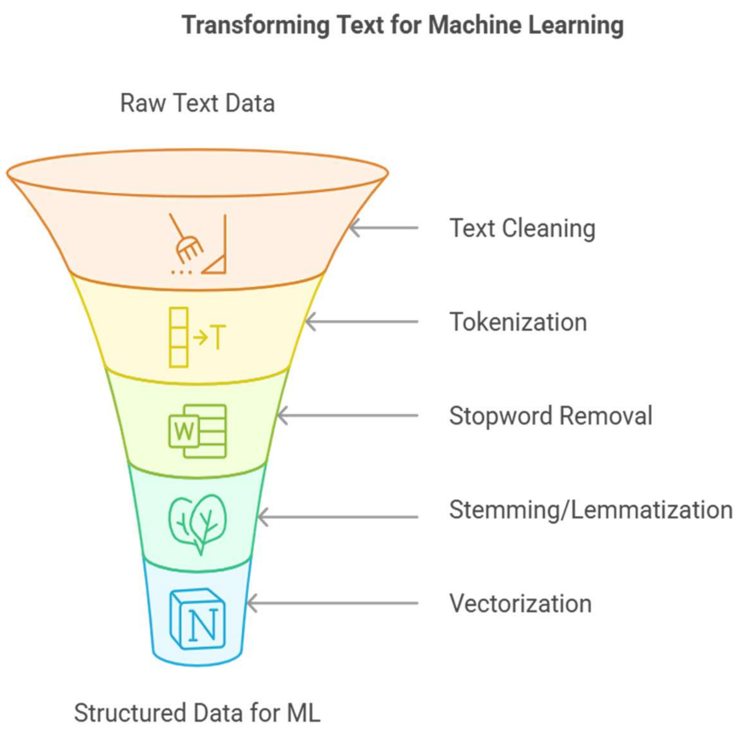

Fig 3. NLP Preprocessing Pipeline

Stage	**Description**
Text Cleaning	Removing noise such as punctuation, special characters, and symbols.
Tokenization	Splitting the text into individual words or phrases.
Stopword Removal	Removing frequently used words (e.g., "the", "is").
Stemming and Lemmatization	Reducing words to their root form.

Vectorization	Converting the text into numerical form using methods like Bag of Words (BoW), Term Frequency-Inverse Document Frequency (TF-IDF), or word embeddings.

These steps include input data compression but the retention of the major information for classification.

NLP Preprocessing Pipeline

Preprocessing in NLP is a very important step because it includes the cleansing and conversion of raw text to be in a format understood by the model system. It aims to reduce the noise, standardizing the text and producing sensible input for the model. The following steps can be included in a usual NLP preprocessing pipeline:

1. Text Cleaning:

Removing unwanted characters or symbols and formatting problems from the text.

Example:
Original text: "I'm so happy!! This is the BEST day of my life!!! #happy #blessed"
After cleaning: "I'm so happy This is the BEST day of my life"

Common techniques:

- Remove special characters (#, @, !, ?)
- Convert to lowercase
- Remove HTML tags, URLs, and emojis

2. Tokenization:

This refers to splitting the strings into units which may be either words or sentences.

Example:
Text: "I'm so happy"
Tokens: ["I'm", "so", "happy"]

Techniques:

- Word based tokenization
- Sentence based tokenization

3. Stopword Removal

Common little words such as "the," "is," "and" constitute the "noise" and thus do not contribute to the meaning of the text. The fact that they have no meaning will tend to reduce the dimensionality of the data.

Example:
Tokens: ["I'm", "so", "happy"] → After removing stopwords: ["happy"]

4. Lemmatization

Lemmatization reduces the words to their root form while considering the context and meaning.

Example:
"running" → "run", *"better" → "good"*

5. Vectorization

Convert the text into numerical format so that it can be fed into machine learning models. Common methods includes:

- **Bag of Words (BoW)** – Count based on representation of words
- **TF-IDF** – Measures the importance of words based on their frequency
- **Word Embeddings** – Context based representation of words (e.g., Word2Vec, GloVe)

Example of Pipeline Working:
Input:
"I'm so happy!! This is the BEST day of my life!!! #happy #blessed"

Step 1: Clean the text → "I'm so happy This is the BEST day of my life"
Step 2: Tokenize → ["I'm", "so", "happy", "This", "is", "the", "BEST", "day", "of", "my", "life"]
Step 3: Remove stopwords → ["happy", "BEST", "day", "life"]
Step 4: Lemmatize → ["happy", "best", "day", "life"]
Step 5: Vectorize → [1, 0, 1, 1] *(example BoW vector)*

2.2 Sequence Modeling with LSTM

Traditional types of neural networks fail to handle sequential data quite well as they lack memory elements. RNN solves this problem by introducing hidden states enabling the information to be stored in the model over the time steps. Yet, there is a problem in RNN, which is described as the vanishing gradient, where during backpropagation, the gradients vanish leading to little or null learning on very long sequences.

Here is what the LSTM network does:

- **Cell State** – The memory semantics that carry the information over time steps.
- **Input Gate** – Controls new things to include in the cell state.
- **Forget Gate** – Decides what information has to be discarded in the cell state.
- **Output Gate** – Decides upon what information would be exposed as output, based on the current cell state.

Therefore, LSTM networks are very efficient for sentiment analysis, extending a long memory of whole sequences that help in context and sentiment understanding in a complete sentence.

2.3 Bidirectional LSTM

Bidirectional long short term memory (BiLSTM) which is an extension of the original LSTM can process the input sequence in both the forward as well as the backward way to capture the context at both ends. This provides the model the capability to understand the usage of a particular word or phrase in context to the past and future making it more effective for contextual understanding of the text. In a BiLSTM configuration, the output from each time step becomes as follows:

$$h_t = h_t^{\text{forward}} + h_t^{\text{backward}}$$

where:

- h_t^{forward} = Hidden state from the forward LSTM
- h_t^{backward} = Hidden state from the backward LSTM

This combined hidden state enhances the model's ability to understand complex linguistic structures such as sarcasm and negation.

3. Data Preprocessing and Feature Engineering

3.1 Tokenization and Padding

Text data needs to be converted into numerical format before feeding it to a neural network.

- Tokenization means dividing text into each individual word and assigning a unique integer index to each such word.
- Padding adds zeroes to make each of these sequences equal size with the longest.

Example Code:

```
tokenizer = Tokenizer(num_words=10000, oov_token="<OOV>")
tokenizer.fit_on_texts(X_train)

# Convert to sequences
train_sequences = tokenizer.texts_to_sequences(X_train)

# Pad sequences to fixed length
train_padded = pad_sequences(train_sequences, maxlen=100, padding='post')
```

3.2 Word Embeddings

Word embeddings put the words in a dense vector space where semantically similar words are found together. The addition of pre-trained embeddings like Word2Vec or GloVe gives better model performance by incorporating various levels of semantic relationships.

Example of initializing an embedding layer in Code:

```
model.add(Embedding(input_dim=10000, output_dim=128, input_length=100))
```

This creates a 128-dimensional vector for each word in the vocabulary.

4. Model Architecture

The components of the sentiment analysis architecture are:

- **Embedding Layer** – Converts tokens to word vectors.
- **Bidirectional LSTM** – Processes both forward and backward contexts.
- **Dropout Layer** – To evade overfitting, this layer randomly turns off calculative neurons.
- **Dense Layer** – Produces a single value varying between 0 and 1 via the sigmoid function.

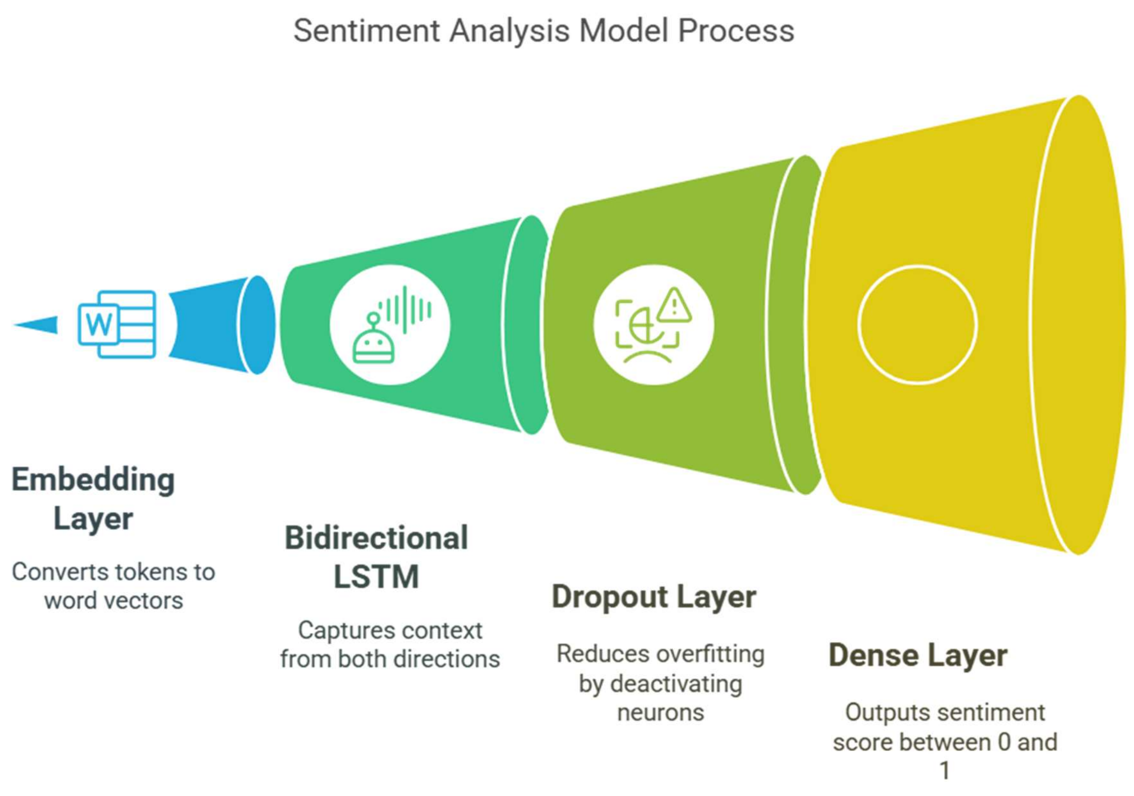
Sentiment Analysis Model Process
Embedding Layer
Converts tokens to word vectors
Bidirectional LSTM
Captures context from both directions
Dropout Layer
Reduces overfitting by deactivating neurons
Dense Layer
Outputs sentiment score between 0 and 1

Fig 4. Sentiment Analysis Model Process

Example Code:

```
model = Sequential()
model.add(Embedding(10000, 128, input_length=100))
model.add(Bidirectional(LSTM(64,return_sequences=True,kernel_regularizer=l2(0.01))))
model.add(Dropout(0.6))
model.add(LSTM(16))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))
```

The model is compiled using **binary cross-entropy** loss and the Adam optimizer:

```
model.compile(loss='binary_crossentropy',optimizer=Adam(learning_rate=0.0005),
metrics=['accuracy'])
```

5. Evaluating the Model

After training the model, it’s essential to evaluate how well it performs on both the training and validation data. Let’s analyze the results step-by-step:

1. Accuracy and Loss Trends

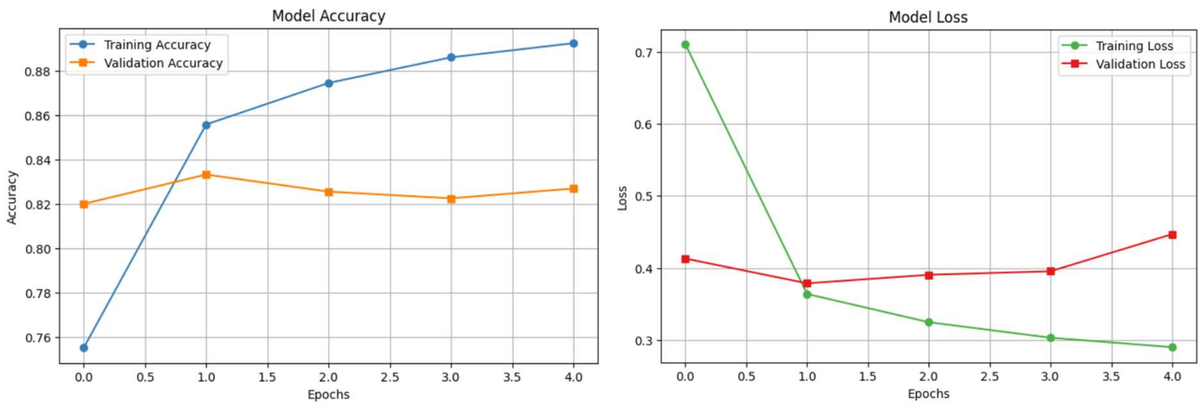

Fig 5. Accuracy and Loss of LSTM Model

From the graphs given below, it can be stated about the accuracy and loss for both training and validation within five epochs:

- For the training accuracy, it increases progressively to around 90% indicating that the model is learning well from the training data.
- Nevertheless, the validation accuracy is at a steady 83%, which means the model is somewhat able to memorize and learn the underlying patterns from the training set, but is not able to generalize very well to unseen data.
- Training loss continues to reduce, and the validation loss appears to stall after the first epoch, indicating the possibility that the model is starting to overfit the training data.

➢ **Insight:** If the training loss decreases while validation loss is upheld higher or increasing, it indicates overfitting, for which one could consider using some regularisation/dropout or increasing the size of the dataset.

2. Confusion Matrix and Classification Report

Now let us study the confusion matrix and classification report to understand how well the model performs for each class:

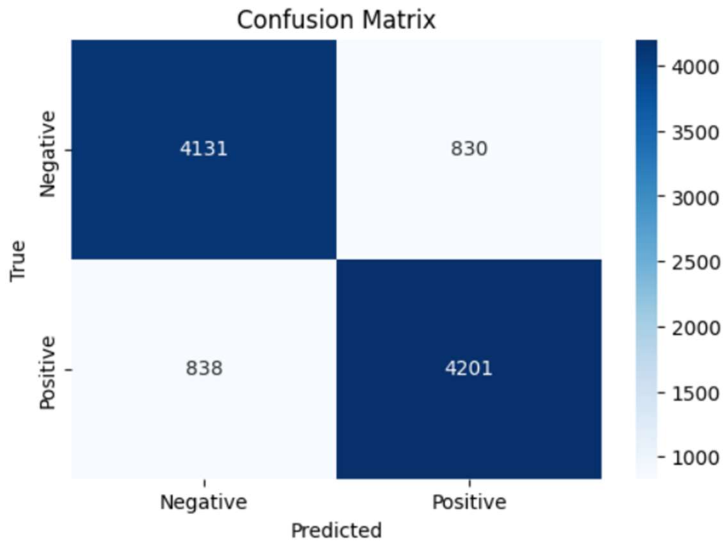

Fig 6. Confusion Matrix of Prediction

- The confusion matrix shows that the model has got most of the positive and negative right.
- Approx, 1,038 negative samples are wrongly classified as positive, and 618 positive samples are wrongly classified as negative.
- The overall accuracy is 84%, and the model has better precision for the negative class (86%) and it has recall for the positive class (88%).

➢ **Precision V/s Recall:** If the precision value of the negative class is high, then model is able to avoid false positives very well. If the recall of the positive class is high, it means it captures most positive cases, even if that means it is misclassifying a few of them.

➢ **What Can Be Done to Improve the model?**

There is overfitting problem which appears to exist as suggested by the accuracy gap between training and validation. To solve it, we can:

- Further use different regularization (L2 or dropout).
- Increase dataset size or apply data augmentation.
- Change the learning rate or change the early stopping to reduce overfitting.

To conclude, in NLP, the best method relies on the kind of text data one has and what task one is concerned with. For example, for sentiment analysis, good methods are always word embeddings such as Word2Vec or GloVe; tasks like language translation or named entity recognition (NER) are perfectly fit for LSTM or GRU models. Justifiably, transformer-based models such as BERT and GPT vastly enhanced NLP by learning contextualization, while older methods such as Bag of Words (BoW) or TF-IDF still provide solid performance for simpler tasks such as text classification. The bottom line is really to understand your data, and try out different methods to identify what gives best performance for your specific task.

References

- Goldberg, Y. (2017). **Neural Network Methods for Natural Language Processing**. *Synthesis Lectures on Human Language Technologies*. Link
- Hochreiter, S., & Schmidhuber, J. (1997). **Long Short-Term Memory**. *Neural Computation, 9(8), 1735-1780*. Link
- Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). **BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding**. *arXiv preprint*. Link
- Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). **Distributed Representations of Words and Phrases and their Compositionality**. *Advances in Neural Information Processing Systems (NeurIPS)*. Link
- Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A. Y., & Potts, C. (2013). **Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank**. *Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Link