Next word Generation of text using Long Short Term Memory (LSTM)

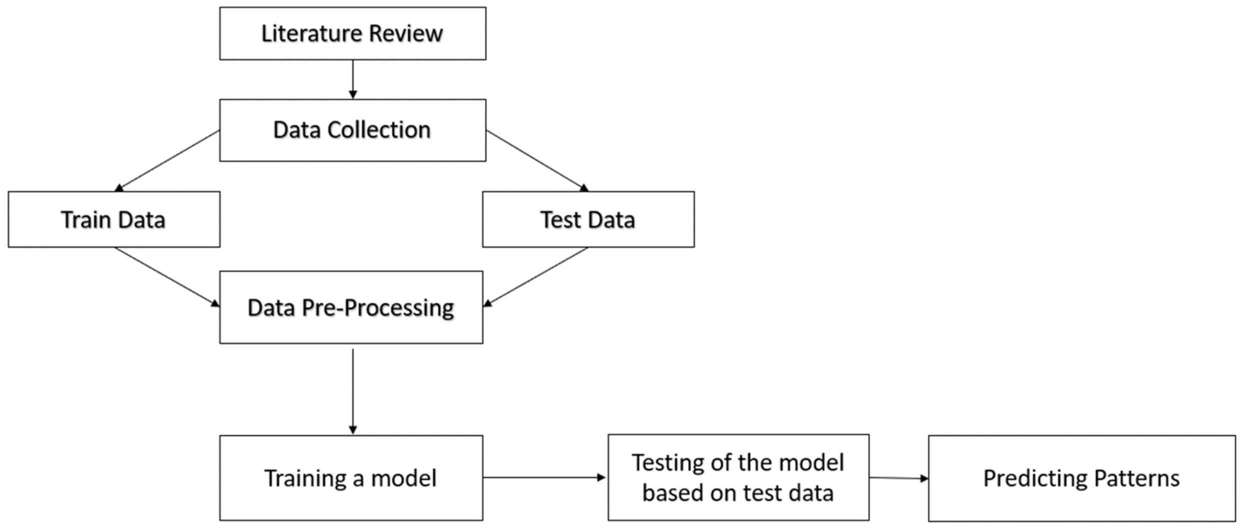
ABSTRACT:
In this project, we are building a next word prediction model with the help of Long Short Term Memory (LSTM). Next word generation of text is very useful for the users who are using modern technology devices in daily life. It helps to fasten their work and also reduce the errors made by the user while messaging. In this project, the proposed approach applies a text mining approach and a LSTM model to predict the next word. Furthermore, we evaluate the model accuracy and loss by changing the hyperparameters like epochs and learning rate.

Keywords: LSTM, Prediction, NLP, Text Generation

INTRODUCTION:
As the usage of technology is increasing, the efforts made by humans are decreasing. It is very hard for the human to find the missing texts in the document and to fasten the given work in their jobs. Machine learning and predictive analytics are becoming increasingly popular in the field of text mining. This project gives a detailed description of the implementation of an algorithm that is able to predict the next word, thereby improving speed at the workplace and also to reduce the errors. The suggested framework for text analysis uses a text mining method and a deep learning technique called LSTM. Text mining helps in extracting essential factors and patterns from enormous amounts of data that might otherwise go unnoticed using standard approaches. We train the available data using LSTM and predict future words.

DATASET:
We are taking a text file as input. The text file is taken from the random collection of the books found on the internet.

METHOD:
Traditional data mining approaches such as statistical analyses are less effective and efficient for text analysis, so machine learning models are favored. The methods used for analysis are depicted in the flowchart shown below.

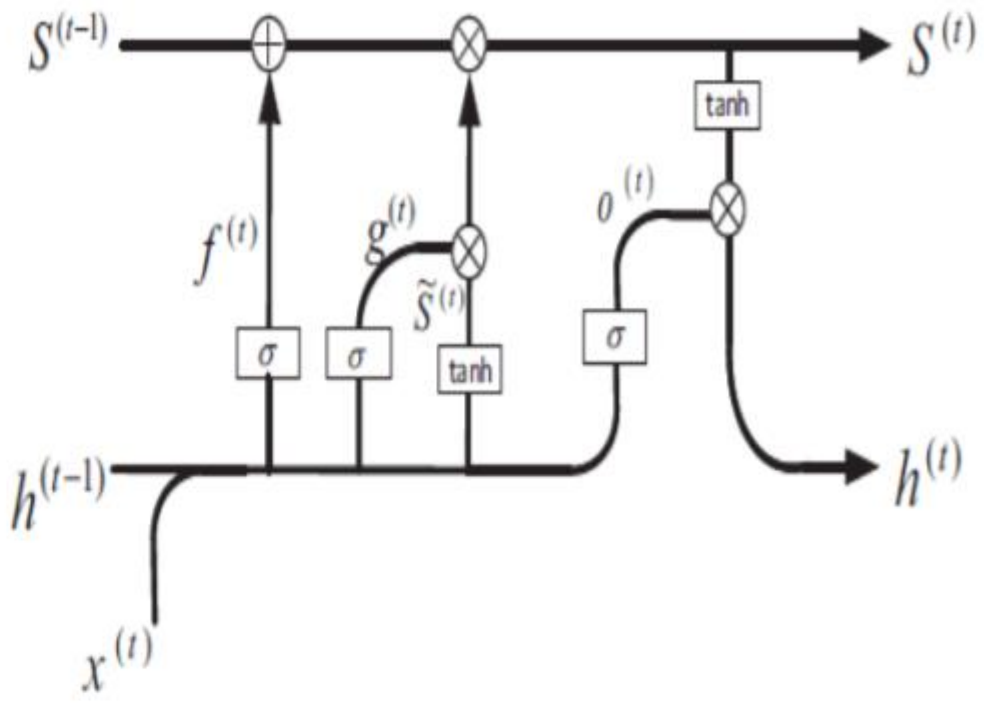


Methodological Flow Chart

The LSTM network is a recurrent neural network that is trained over time using Backpropagation to avoid the vanishing gradient problem. We predict the words by using this technique with the available data.

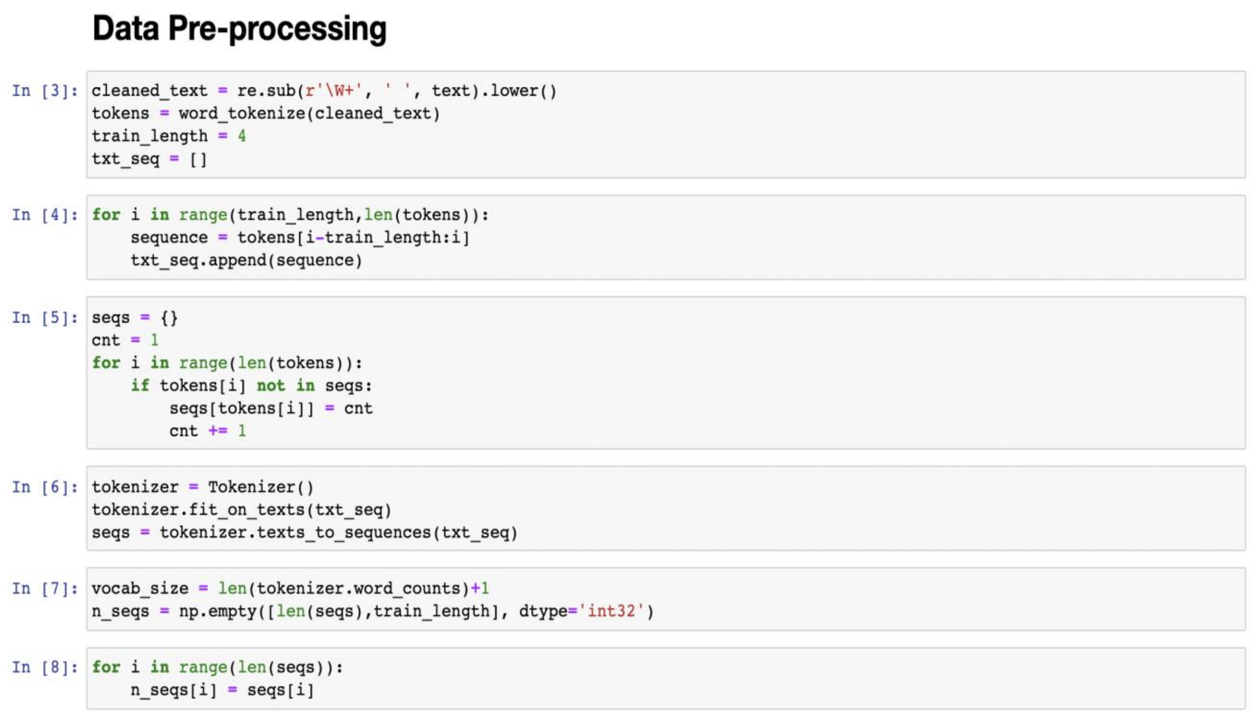
LONG SHORT TERM MEMORY:

The main benefit of Recurrent Neural Networks (RNNs) is that they use context sensitive data to map processes between input and output sequences. However, up to a certain point, RNNs can obtain information. With each recurrence of the network loop, this constraint reduces the influence of the hidden layer's input on the network's output. So, to overcome this limitation we use Long and Short Time Memory (LSTM). Each LSTM cell unit also has one internal self-loop and the RNN external cell. Each cell has three gates namely the input gate, the output gate, and the forgotten gate. The output of the sigmoid layer has the elements range from 0 to 1. If the complete data is passed then it is indicated by 1 and if no information is passed then it is indicated by 0. The figure below illustrates the LSTM cell structure.



LSTM Cell Structure

DATA PREPROCESSING:



Data Pre-processing

```
In [3]: cleaned_text = re.sub(r'\W+', ' ', text).lower()
        tokens = word_tokenize(cleaned_text)
        train_length = 4
        txt_seq = []
```

```
In [4]: for i in range(train_length,len(tokens)):
            sequence = tokens[i-train_length:i]
            txt_seq.append(sequence)
```

```
In [5]: seqs = {}
        cnt = 1
        for i in range(len(tokens)):
            if tokens[i] not in seqs:
                seqs[tokens[i]] = cnt
                cnt += 1
```

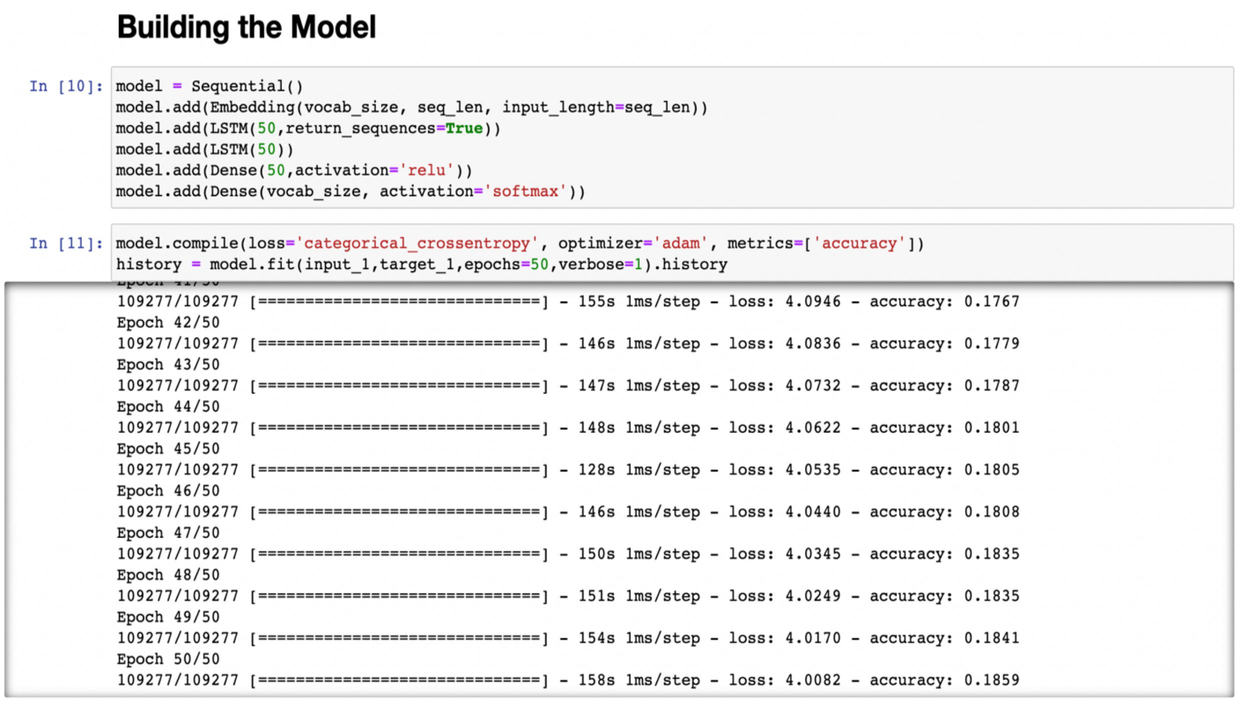
```
In [6]: tokenizer = Tokenizer()
        tokenizer.fit_on_texts(txt_seq)
        seqs = tokenizer.texts_to_sequences(txt_seq)
```

```
In [7]: vocab_size = len(tokenizer.word_counts)+1
        n_seqs = np.empty([len(seqs),train_length], dtype='int32')
```

```
In [8]: for i in range(len(seqs)):
            n_seqs[i] = seqs[i]
```

Data preparation is an important stage in the data mining process. The quality of data has the greatest impact on the quality of outcomes. We use tokenizers and feature engineering. Tokenizers split the text into words, where we break human readable text into machine readable components. Feature engineering refers to the process of leveraging domain expertise to choose and convert the most important variables from raw data.
First we create features dictionary sequences, then we encode it into the integer form with the help of the Tokenizer, after that we split the sequences into the inputs and output, finally we convert our output labels into one-hot vectors.

MODEL BUILDING:



Building the Model

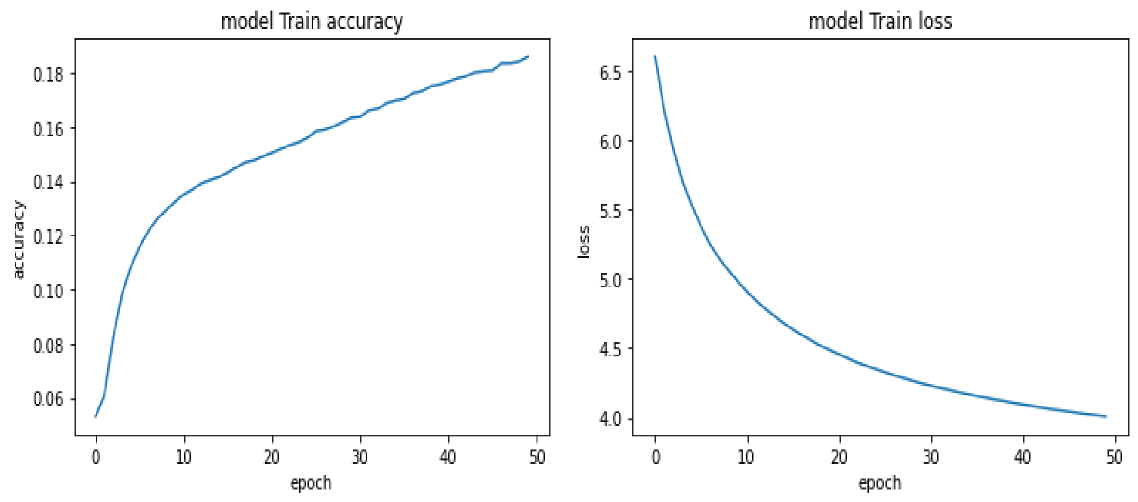
```
In [10]: model = Sequential()
         model.add(Embedding(vocab_size, seq_len, input_length=seq_len))
         model.add(LSTM(50,return_sequences=True))
         model.add(LSTM(50))
         model.add(Dense(50,activation='relu'))
         model.add(Dense(vocab_size, activation='softmax'))
```

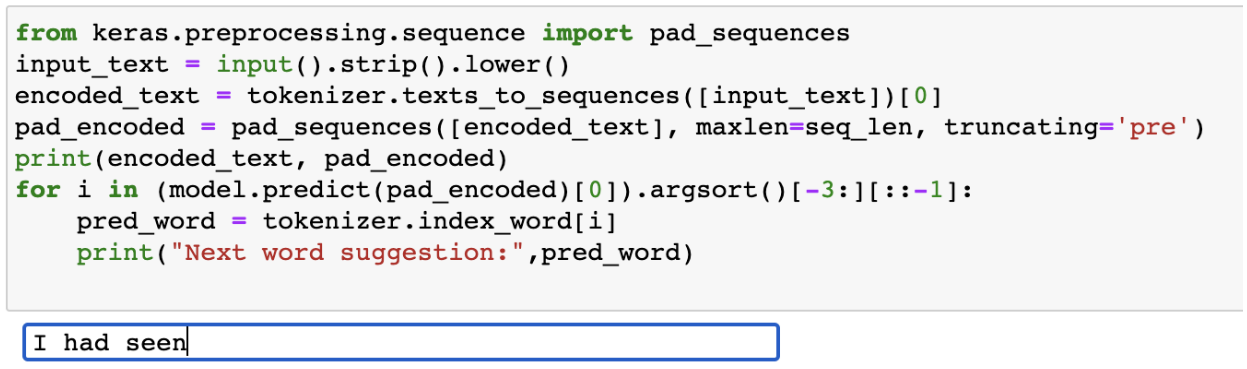
```
In [11]: model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
         history = model.fit(input_1,target_1,epochs=50,verbose=1).history
```

```
109277/109277 [==============================] - 155s 1ms/step - loss: 4.0946 - accuracy: 0.1767
Epoch 42/50
109277/109277 [==============================] - 146s 1ms/step - loss: 4.0836 - accuracy: 0.1779
Epoch 43/50
109277/109277 [==============================] - 147s 1ms/step - loss: 4.0732 - accuracy: 0.1787
Epoch 44/50
109277/109277 [==============================] - 148s 1ms/step - loss: 4.0622 - accuracy: 0.1801
Epoch 45/50
109277/109277 [==============================] - 128s 1ms/step - loss: 4.0535 - accuracy: 0.1805
Epoch 46/50
109277/109277 [==============================] - 146s 1ms/step - loss: 4.0440 - accuracy: 0.1808
Epoch 47/50
109277/109277 [==============================] - 150s 1ms/step - loss: 4.0345 - accuracy: 0.1835
Epoch 48/50
109277/109277 [==============================] - 151s 1ms/step - loss: 4.0249 - accuracy: 0.1835
Epoch 49/50
109277/109277 [==============================] - 154s 1ms/step - loss: 4.0170 - accuracy: 0.1841
Epoch 50/50
109277/109277 [==============================] - 158s 1ms/step - loss: 4.0082 - accuracy: 0.1859
```

we trained our sequential model that has 5 layers: An Embedding layer, two LSTM layers, and two Dense layers with 50 epochs and achieved training accuracy of 18%.

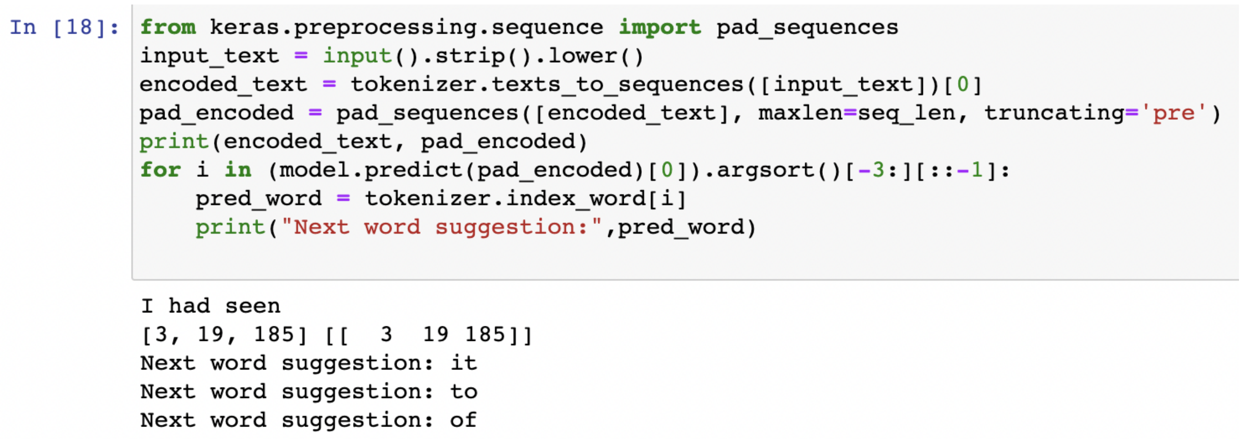
RESULTS:





```
from keras.preprocessing.sequence import pad_sequences
input_text = input().strip().lower()
encoded_text = tokenizer.texts_to_sequences([input_text])[0]
pad_encoded = pad_sequences([encoded_text], maxlen=seq_len, truncating='pre')
print(encoded_text, pad_encoded)
for i in (model.predict(pad_encoded)[0]).argsort()[-3:][::-1]:
    pred_word = tokenizer.index_word[i]
    print("Next word suggestion:",pred_word)
```

```
I had seen
```



```
In [18]: from keras.preprocessing.sequence import pad_sequences
         input_text = input().strip().lower()
         encoded_text = tokenizer.texts_to_sequences([input_text])[0]
         pad_encoded = pad_sequences([encoded_text], maxlen=seq_len, truncating='pre')
         print(encoded_text, pad_encoded)
         for i in (model.predict(pad_encoded)[0]).argsort()[-3:][::-1]:
             pred_word = tokenizer.index_word[i]
             print("Next word suggestion:",pred_word)

         I had seen
         [3, 19, 185] [[  3  19 185]]
         Next word suggestion: it
         Next word suggestion: to
         Next word suggestion: of
```

LSTM model with Embedding layer, two LSTM layers and tow dense layers with 50 epochs gives accuracy of 18% and the accuracy plot shows that the model's performance or accuracy can be improved by increasing the number of epochs.

CONCLUSION AND FUTURE WORK:

During the course of this project, I worked with real-time application data and built a model which could predict the next word. We used the LSTM model to predict the next word if given a sentence. In future we are planning to improve upon this LSTM using attention techniques to help the model remember the data far into the past history of words. We could also try to train recently introduced BERT style models which were based on transformers which would improve the word prediction accuracy.

REFERENCES:

1. https://www.hindawi.com/journals/wcmc/2021/5886119/
[1] Yuqi Liu, Zhongfeng Su, Hang Li, and Yulai Zhang. An lstm based classification method for time series trend forecasting. In 2019 14th IEEE Conference on Industrial Electronics and Applications (ICIEA), pages 402–406, 2019.
2. https://www.analyticsvidhya.com/blog/2021/08/predict-the-next-word-of-your-text-using-long-short-term-memory-lstm/
3. https://towardsdatascience.com/dont-overfit-how-to-prevent-overfitting-in-your-deep-learning-models-63274e552323
4. https://towardsdatascience.com/next-word-prediction-with-nlp-and-deep-learning-48b9fe0a17bf