Text Summarization using BBC News Articles

Lasya Manthripragada, Nithish Goud Podeti, Saketh Dathrika, Varsha Reddy Anugu
Khoury college of Computer Sciences
Northeastern University
Boston, MA 02115
manthripragada.l@northeastern.edu, podeti.n@northeastern.edu, dathrika.s@northeastern.edu,anugu.v@northeastern.edu

Abstract

In this project, we implemented text summarization and topic modelling using NLP and Deep learning using BBC news summary dataset from Kaggle. The amount of text data available has exploded from a variety of sources. This volume of material has a wealth of information and knowledge that must be adequately summarized to be useful. In this review, we took different approaches to text summarization and topic modelling (LSTM, LDA) and analyzed the models we created. Our experiments suggest that if we build a BERT model, it will increase the performance by finding advance text summarization.

1. Introduction

One of the most common forms of communication is writing. Due to the enormous number of languages spoken around the world, there are many different styles of writing. Thousands of texts are written every day by people all around the world. It is necessary to read the documents entirely in order to get the core idea and basic information that they provide to the readers. Many individuals are curious about how texts convey desired or vital information through sentences, or, in other words, how many of those sentences are required to convey the main subject. In any document, the number of unnecessary sentences might sometimes outnumber the amount of desired information. As a result, readers are forced to read the entire text only to figure out what the post is about or what is meant to be conveyed, which wastes a lot of time. The automatic text summarizing topic has received a lot of interest in order to extract the primary theme or desired information from texts and solve these problems.

Automatic summary techniques make it simple for readers to grasp the essential point of a book. They are able to eliminate unnecessary data as a result of this. As a result, while reading time is reduced, text importance is increased. As a result, information is more easily accessible.

Depending on the application approaches and methods, automatic summarization systems can be classified as extractive or abstractive. Because abstractive summarization systems still perform poorly and have less precise information than extractive summarization systems, this research has concentrated on information extraction. This is because extractive algorithms choose information directly from the primary text rather than attempting to generate new information on their own. Even if extractive summarizing algorithms aren't ideal, they provide readers with a viewpoint on the original material.

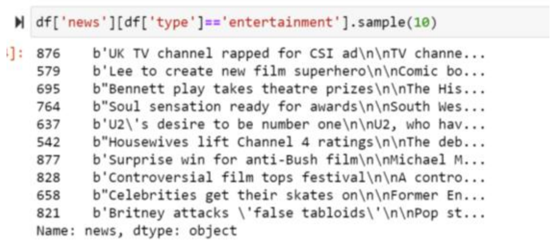
2. Method

We reimplemented the LSTM from Colah's blog [1] and tuned it accordingly. W built a 3-Stack LSTM for text summarization. The tools which we used are Scikit-learn [2] and TensorFlow [3], as well as matplotlib and seaborn for some visualization. We used LDA from Genism package [4] and 3- stacked LSTM [5] to summarize the news article. Stacked LSTM has multiple layers of LSTM stacked on top of each other. This leads to a better representation of the sequence. We visualized the data, and test whether the summarization for each news article is accurate or not.

2.1 Dataset and Data Pre-processing

The Dataset is taken from Kaggle: https://www.kaggle.com/pariza/bbc-news-summary .

This dataset is used for extractive text summarization has business, political, sports, entertainment, and tech news articles of BBC from 2004 to 2005 in the News Articles folder. For each article, five summaries are provided in the Summaries folder. The first clause of the text of articles is the respective title. Figure 1 shows a sample of the dataset.

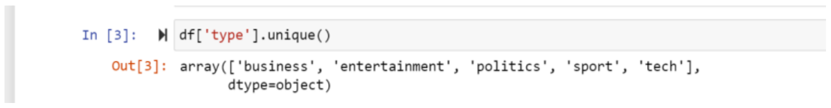


```
df['news'][df['type']=='entertainment'].sample(10)
```

```
]: 876    b'UK TV channel rapped for CSI ad\n\nTV channe...
   579    b'Lee to create new film superhero\n\nComic bo...
   695    b"Bennett play takes theatre prizes\n\nThe His...
   764    b"Soul sensation ready for awards\n\nSouth Wes...
   637    b'U2\'s desire to be number one\n\nU2, who hav...
   542    b"Housewives lift Channel 4 ratings\n\nThe deb...
   877    b'Surprise win for anti-Bush film\n\nMichael M...
   828    b'Controversial film tops festival\n\nA contro...
   658    b"Celebrities get their skates on\n\nFormer En...
   821    b'Britney attacks \'false tabloids\'\n\nPop st...
   Name: news, dtype: object
```

Figure 1 Sample dataset

It also had topics for news articles such as Business, technology etc., The five unique topics for this dataset are shown below in figure 2.



```
In [3]:  df['type'].unique()

Out[3]:  array(['business', 'entertainment', 'politics', 'sport', 'tech'],
               dtype=object)
```

Figure 2: Unique topics

As we can notice in figure 1, there is unnecessary information like '\s' or emails and other related symbols. So, we did some data pre-processing like We did basic data cleaning like

1. Removing emails
2. Removing new lines
3. Removing quotes
4. We tokenized the data and lemmatized it.
5. We also vectorized the data using countVectorizer.

2.2 3- Stacked LSTM:

Long short-term memory (LSTM) is a deep learning architecture based on artificial recurrent neural networks (RNNs). LSTM features feedback connections, unlike normal feedforward neural networks. The Stacked LSTM is an expansion of this model that includes multiple hidden LSTM layers with several memory cells in each layer. For difficult sequence prediction issues, they have become a reliable approach. The 3-Stacked LSTM uses encoder-decoder architecture
.

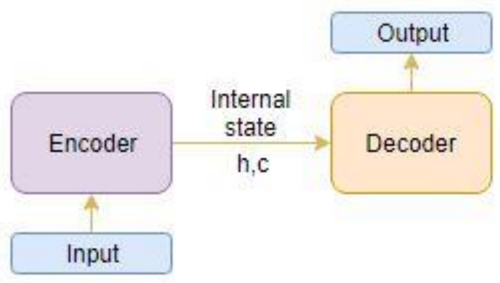


Figure 3: LSTM Architecture

The Encoder-Decoder Architecture can be set in 2 phases:

- Training Phase
- Inference Phase

i. Training Phase:

We will first set up the encoder and decoder during the training phase. The model will then be trained to anticipate the target sequence with one timestep offset. The entire input sequence is read by an encoder with one word being sent into it at each timestep. From Fig.4, we can see that the entire data is then processed at each timestep, and this information is captured.

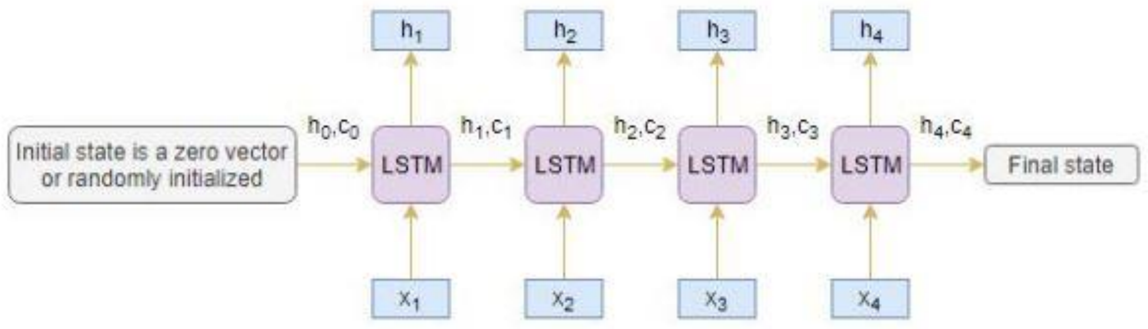


Figure 4: Training Phase of Encoder.

In the training phase of the Decoder, LSTM network that reads the entire target sequence word for word and predicts a sequence that is one timestep off. Given the previous word, the decoder is trained to anticipate the next word in the sequence.

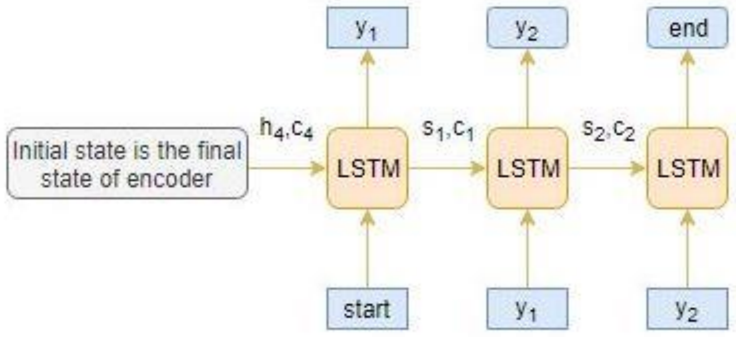


Figure 5: Training Phase of Decoder

<start> and <**end>** are used as tokens which are added to the target sequence before feeding it into the decoder. The target sequence is unknown while decoding the test sequence. So, we start predicting by passing the first word into the decoder which is the <**start>** token. And the <**end>** token signals at the end of the sentence.

ii. Inference Phase:

After training, the model is put to the test on new source sequences with unknown target sequences. To decode a test sequence, we must first build up the inference architecture.

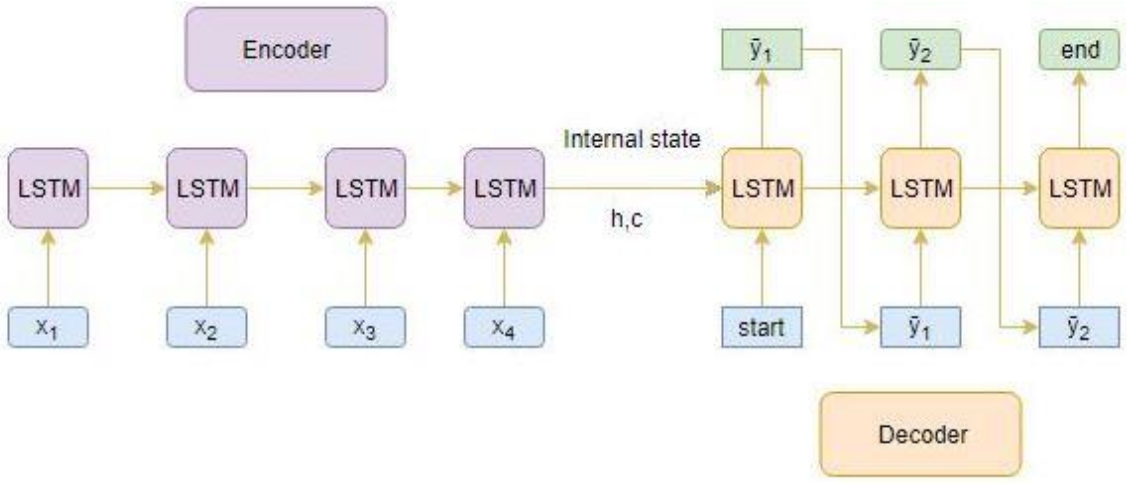


Figure 6: Inference Phase

From Fig.6 we can get the steps to decode the test sequence:

1. Entire Input sequence is encoded and initialize it to the decoder with internal states of the encoder.
2. Pass <start> token as an input to the decoder.
3. Run the decoder for one timestep with the internal states.
4. The output will be the probability for the next word. The word with the maximum probability will be selected.
5. Pass the sampled word as an input to the decoder in the next timestep and update the internal states

with the current time step.

6. Repeat the steps 3 – 5 until we generate <end> token or hit the maximum length of the target sequence.

iii. Attention Mechanism:

This architecture is only good for short sequences since the decoder is looking at the entire input sequence for the prediction. It is difficult for the encoder to memorize long sequences into a fixed length vector. To overcome this problem attention mechanism is used. It aims to predict a word by looking at a few main parts of the sequence rather than the whole sequence.
There are 2 different classes of attention mechanism depending on the way the attended context vector is derived:
Global Attention (GA)
Local Attention (LA)

Global Attention uses all the hidden states of the encoder for deriving the attended context vector whereas LA uses a few hidden states of the encoder are considered for deriving the attended context vector. Here we used global attention to train the model.

iv. Metrics for summarization:

Rouge is the metric which is a package used to evaluate automatic summarization.
There are many rouge metrics such as:

ROUGE-N: Overlap of n-grams between the correct and predicted summaries.
ROUGE-1 refers to the overlap of unigram (each word) between the original and predicted summaries.
ROUGE-2: It is the overlap of bigrams between the system and reference summaries.
We used Rouge-1 metric for our model.
The formula of it is given by:

$$\frac{number\_of\_overlapping\_words}{total\_words\_in\_reference\_summary}$$

2.3 LDA

"LDA (*short for Latent Dirichlet Allocation*) is an unsupervised machine-learning model that takes documents as input and finds topics as output. The model also says in what percentage each document talks about each topic." [6]

There are 3 main parameters of the model:

a. the number of topics
b. the number of words per topic
c. the number of topics per document

There is a nice way to visualize the LDA model you built using the package *pyLDAvis* [7]. This visualization allows you to compare topics on two reduced dimensions and observe the distribution of words in topics. We will talk more about this in the results section.
Working of LDA more clearly is given in Figure 7, where the words frequently repeated in documents are clustered and thus LDA groups different topics.

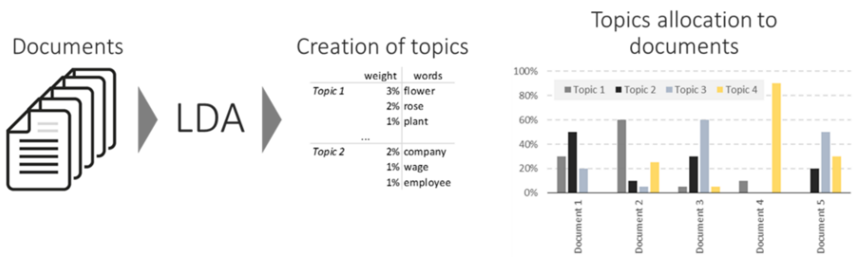


Figure 7: Working of LDA.

- LDA Input:
  - N no. of documents
  - Each of these documents have N no. of words
  - Hyperparameters- alpha and beta
  - All of which needs to pass through LDA
- LDA Output:
  - K no. of topics
  - Document to topic distribution and Topic-term distribution

These are heavily referred from the article [6
], where the author takes about implementing LDA for topic modelling and visualizing the clustering. To determine whether or whether your model makes sense, you must first view the topics. In the case of K-Means, LDA converges, and the model makes mathematical sense, but this does not imply that it makes human sense. Cleaning your data involves removing stop words that appear too frequently in your themes and re-running your model. Your themes will improve if you keep only nouns and verbs, remove templates from texts, and test different cleaning procedures repeatedly.
The author also talked about different advantages and disadvantages in LDA.

Advantages:
- It is fast
- It is intuitive
- It can predict topics for new unseen documents.

Dis-Advantages:
- It requires Fine tuning.
  It needs human interpretation.

3 Results

For LDA, there are three important metrics to measure if our model is functioning properly- 1. Topics should be interpretable. 2. Topics are unique. 3. All the documents should be represented. According to Fig.8, word distribution is not overlapping, and there is a clear distinction among topics.

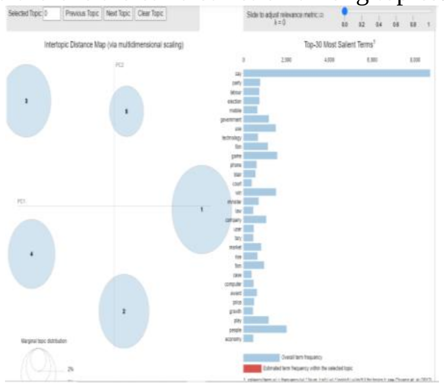

Selected Topic: 0
Previous Topic
Next Topic
Clear Topic
Intertopic Distance Map (via multidimensional scaling)
Top-30 Most Salient Terms
PC1
PC2
Marginal topic distribution

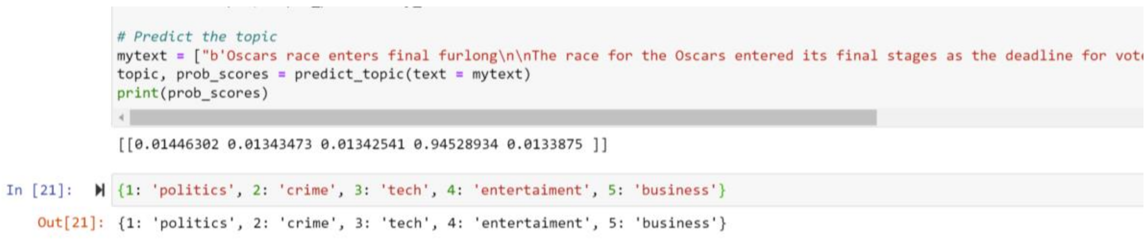
Figure – 8 LDA topic clustering.

Table 1 shows the results of LDA model. *Log-likelihood function* is a logarithmic transformation of the likelihood function, Likelihood - measure of how plausible model parameters are given the data Taking a logarithm makes calculations easier All values are negative: when x< 0 log x <0. Perplexity is a measure of model's "surprise" at the data Positive number Smaller values are better.

Metrics of LDA	Value
Log-likelihood	-2616563.631168631
Perplexity	1369.4588046576564

Table 1 Metrics of LDA

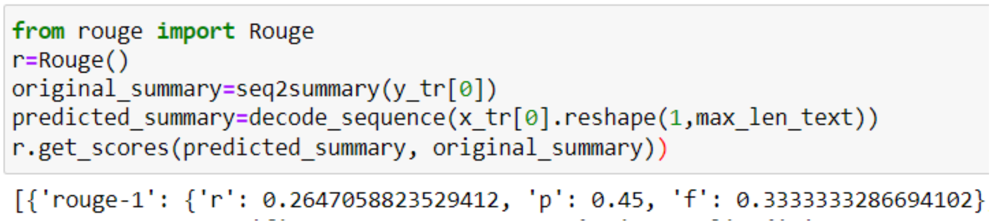
Figure 9 shows the result of an example of how our model works. It was easy classifying the clusters and naming them because it was not overlapping. The example input was about entertainment and the probability for topic 4 is the highest, which is given to be "Entertainment", hence it works out fine.



```
# Predict the topic
mytext = ["b'Oscars race enters final furlong\n\nThe race for the Oscars entered its final stages as the deadline for vot
topic, prob_scores = predict_topic(text = mytext)
print(prob_scores)

[[0.01446302 0.01343473 0.01342541 0.94528934 0.0133875 ]]

In [21]:  {1: 'politics', 2: 'crime', 3: 'tech', 4: 'entertaiment', 5: 'business'}

Out[21]: {1: 'politics', 2: 'crime', 3: 'tech', 4: 'entertaiment', 5: 'business'}
```

Figure 9 An example of topic modelling

Fig 10 shows the rouge-1 metrics for the text summarization. It contains precision, recall and f-score for the predicted summaries of the validation data to the original summaries.



```
from rouge import Rouge
r=Rouge()
original_summary=seq2summary(y_tr[0])
predicted_summary=decode_sequence(x_tr[0].reshape(1,max_len_text))
r.get_scores(predicted_summary, original_summary)

[{'rouge-1': {'r': 0.2647058823529412, 'p': 0.45, 'f': 0.3333333286694102}
```

Figure 10 Rouge-1 metric scores

4 Discussion

The goal of this project is to summarize the given larger text into smaller pieces using extractive text summarization. We have used unsupervised machine learning models like 3-stacked LSTM and also LDA model for summarizing the text. Initially, we have used LSTM model which aims to predict a word by looking at a few main parts of the sequence rather than the whole sequence. Although we used GCP to create GPU instance to run the LSTM we couldn't get the desired results because of the large number of parameters required. In addition to it, we have also done topic modelling using Gensim's LDA which aims to convert documents into topics by assigning percentages to each topic.

For our dataset, we can interpret that LDA model performed well by giving the most desired outputs with the metrics log-likelihood and perplexity rather than with metrics used for LSTM model which are f-score, precision and recall. From the results, we can clearly analyze the same as we discussed. While with data with more computational cost, BERT model works better. Unfortunately, due to lack of time and additional GPU resources for projects like this, we could implement only these two models.

Ultimately, the goal of these modelling is to extract text and summarizing it provide primary information to the readers.

The Github link to the project is: https://github.com/lasyamanthri/CS6120

5 Conclusion

We started LSTM for text summarization but could not get the desired results because as we increase the number of parameters in LSTM the computation cost of the model is increasing which requires additional GPU resources to collaborate it. In addition to it, we built a basic topic model using Gensim's LDA and visualize the topics using pyLDAvis.

6 Future scope

Build a BERT (Bidirectional Encoders) Model to get advanced text summarizations. As the model will be large because of the training structure and corpus. It is slow to train because it is big and there are a lot of weights to update. It requires more computation because of its size, which comes at a cost. Hence limited the project using 3 Stacked LSTM and LDA models.

7 Acknowledgement

This project was completed for CS6120 under the instruction of Professor Dr. Uzair Ahmad.

8 References

[1] LSTM- https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[2] Github: https://github.com/aravindpai/How-to-build-own-text-summarizer-using-deep-learning/blob/master/How_to_build_own_text_summarizer_using_deep_learning.

[3] Tensor flow - https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM

[4] LDA MODEL https://radimrehurek.com/gensim/models/ldamodel.html

[5] LSTM Model: https://www.analyticsvidhya.com/blog/2019/06/comprehensive-guide-text-summarization-using-deep-learning-python/

[6] LDA topic modelling- https://towardsdatascience.com/the-complete-guide-for-topics-extraction-in-python-a6aaa6cedbbc

[7] pyLDAvis - https://pyldavis.readthedocs.io/en/latest/readme.html