Project Report
CSCI-645: Advanced Natural Language Processing
Fall 2018

Barathwaaj Parthasarathy (bpartha)
Pravin Sundar (psundar)
Prashanth Thirukkurungudi Sekar (prthiru)
Archish Ramesh Babu (arameshb)

December 12, 2018

1 Project Objective

Living in such fast-paced life, people need a one stop solution to understand what is happening around the word with just a single glance. Providing users with a cluster of words based on news articles and generating headlines based on user's input and previous interest becomes useful.

In this project topics are generated from a given corpus which forms a first look to the user providing cluster of words that is present in the text. Topic Modelling is an unsupervised technique to classify documents based on the topics generated from the models. Topics are generated from the documents through a word wise analysis and its distribution across different documents learnt from the corpus.

Latent Dirichlet Allocation (LDA), Non-negative Matrix Factorization (NMF) and Markov chain model will be used to generate topics from the corpus. Following this, Long Short Term Memory networks (LSTM) is used to generate meaningful headlines depending on the input from the user.

As a long term goal this can be extended to develop a fully autonomous system to generate topics based on any given corpus and eventually generate meaningful sentences based on user's input and previously selected topics.

2 Data

We will be using news headlines published over 15 years sourced from the reputable Australian news source ABC (Australian Broadcasting Corp.).
This data is available in kaggle: https://github.com/Arturus/kaggle-web-traffic

Data Contains:

- publish_date: Date of publishing for the article in yyyyMMdd format
- headline_text: Text of the headline in Ascii , English , lowercase

- Start Date: 2003-02-19 End Date: 2017-12-31
- Total Records: 1,103,665 (For the purpose of this analysis, a random set of 10,000 records are used)

3 Pre - Processing

Before we begin to build the models pre-processing techniques are employed to the data to improve the performance of the models.

- Remove Stopwords
- Lemmatization
- Tf-Idf transformation
- POS tagging

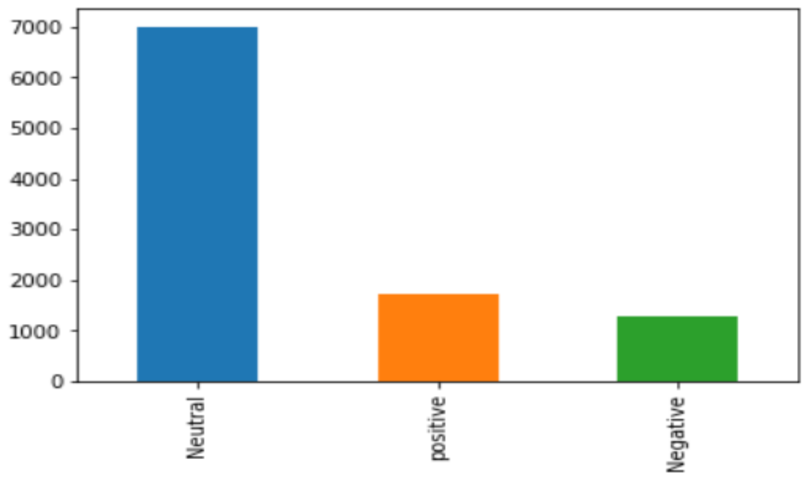
4 EDA

Before moving onto the model building, lets first begin by understanding the data. Since the data is comprised of news article headlines its important to understand the general sentiment of the headlines and the subjectiveness in the headlines because when creating new text it is generally advised to follow the same tone and structure as the original text. The sentiment and subjectivity have been measured by using python's TextBlob package.

4.1 Sentiment

The package automatically tokenises the text, removes stop words and calculates the sentiment of a sentence based on sentiment scores for each of the words. The sentiment score for each word is based on pre-existing corpora. This package effectively deals with negation, modifier words and is also able to capture the extent of the positive or negative sentiment. For our case, three different sentiments were considered, positive if the sentiment score is above 0, negative if the sentiment score is below zero and Neutral if the sentiment score is zero. The below graphs shows distribution of the headlines based on their sentiment. As expected, majority of the sentiment is Neutral followed by positive and negative.

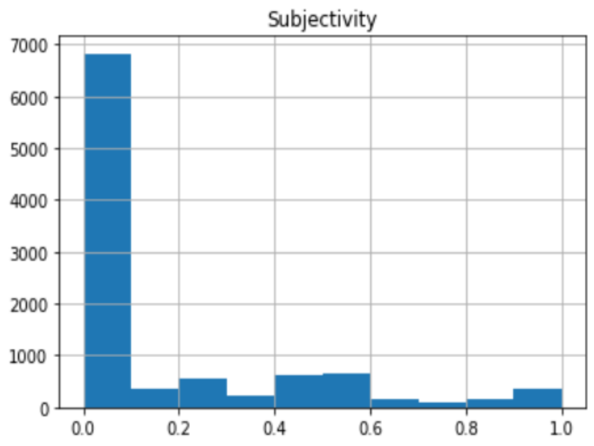
```
In [47]: senti['Sentiment'].value_counts().plot(kind='bar')

Out[47]: <matplotlib.axes._subplots.AxesSubplot at 0x1ba9544a3c8>
```



4.2 Subjectiveness

The subjectiveness score also works along the similar lines as the sentiment measure and follows the same distribution as the Sentiment score.The subjectiveness scores lie between zero and one, zero if the sentence is not subjective and one if the sentence is highly subjective. The below plot shows how the subjectivity is distributed. As expected, majority of the headlines are not subjective and contains factual information.

```
In [69]: senti.hist(column='Subjectivity')

Out[69]: array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000001BC40BE4748>]],
               dtype=object)
```



5 Algorithms

Now that we have completed the EDA phase, we can move on to the model building phase. In the below sections we will first define all the approaches we have selected.

5.1 Latent Dirichlet Allocation

Latent Dirichlet Allocation is the most popular topic modeling technique. LDA is a probabilistic model that updates the probabilities of the words pertaining to a document in an iterative procedure until there is a convergence of the algorithm. This algorithm uses two key probabilities $P(Words|topics)$ and $P(topics|documents)$.
LDA assumes:

- The number of words the document has follows Poisson's distribution
- A topic mixture of the document follows the Dirichlet distribution over a fixed set of K topics
- Picks a topic with multinomial distribution and generates a word using the topic

Post this, LDA uses the words generated to match the topics that is likely to have generated this set.
LDA is a matrix factorization technique. A corpus is factorized into a document-term matrix. For example, let's take a corpus of N documents D1, D2, D3 ... Dn with words W1,W2 .. Wn. The factorized matrix with word frequencies in each document will look like this:

	W1	W2	W3	Wn
D1	0	2	1	3
D2	1	4	0	0
D3	0	2	3	1
Dn	1	1	3	0

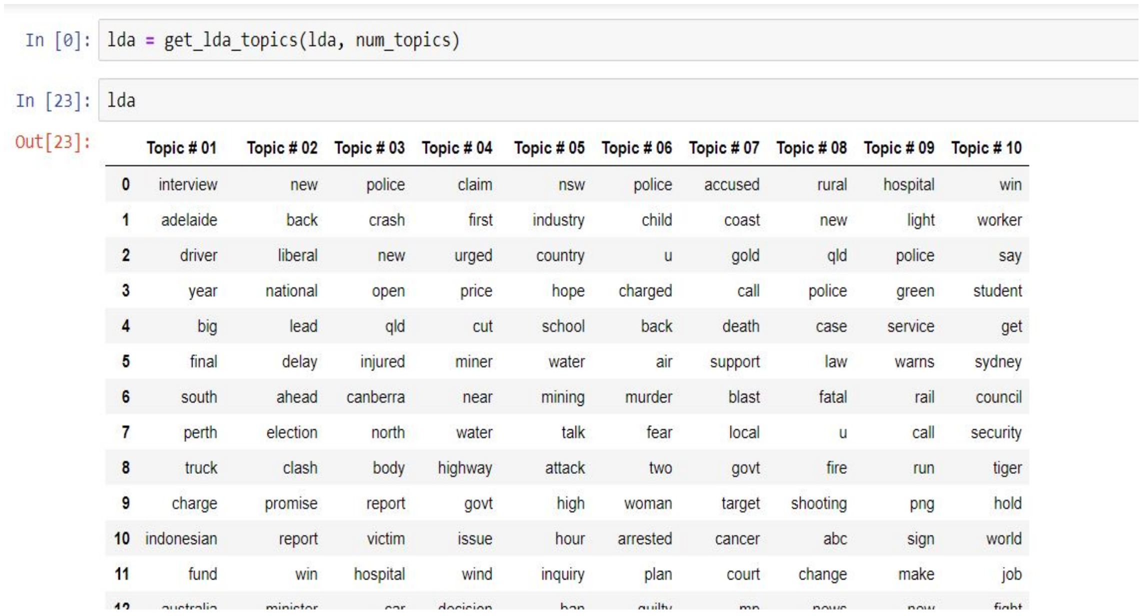
LDA converts this Document-Term Matrix into two lower dimensional matrices. The First is a document-topics matrix and the second is a topic – terms matrix with dimensions (N, K) and (K, M) respectively, where N is the number of documents, K is the number of topics and M is the number of words.

	K1	K2	K3	K
D1	1	0	0	1
D2	1	1	0	0
D3	1	0	0	1
Dn	1	0	1	0

	W1	W2	W3	Wm
K1	0	1	1	1
K2	1	1	1	0
K3	1	0	0	1
K	1	1	0	0

Now, these two matrices provide us the topic-word and document-topic distributions. But this needs to be improved, which is the main aim of LDA. LDA performs an iterative process in order to do this. As mentioned earlier there are two main probabilities calculated, p1 and p2 $P(Words|topics)$ and $P(topics|documents)$.

LDA iterates through each word in a document and tries to change the topic assigned to that word with a probability P which is the product of p1 and p2. After a number of iterations, the document topic and topic term distributions reach the convergence point of LDA.



```
In [0]: lda = get_lda_topics(lda, num_topics)

In [23]: lda
Out[23]:
```

	Topic # 01	Topic # 02	Topic # 03	Topic # 04	Topic # 05	Topic # 06	Topic # 07	Topic # 08	Topic # 09	Topic # 10
0	interview	new	police	claim	nsw	police	accused	rural	hospital	win
1	adelaide	back	crash	first	industry	child	coast	new	light	worker
2	driver	liberal	new	urged	country	u	gold	qld	police	say
3	year	national	open	price	hope	charged	call	police	green	student
4	big	lead	qld	cut	school	back	death	case	service	get
5	final	delay	injured	miner	water	air	support	law	warns	sydney
6	south	ahead	canberra	near	mining	murder	blast	fatal	rail	council
7	perth	election	north	water	talk	fear	local	u	call	security
8	truck	clash	body	highway	attack	two	govt	fire	run	tiger
9	charge	promise	report	govt	high	woman	target	shooting	png	hold
10	indonesian	report	victim	issue	hour	arrested	cancer	abc	sign	world
11	fund	win	hospital	wind	inquiry	plan	court	change	make	job
12	australia	minister	car	decision	ban	guilty	mp	news	now	fight

Above are the topics generated through LDA. As expected, most topics consist of words like Adelaide, Perth, NSW, gold coast, Canberra, etc., related to Australia since ABC news is an Australian organization. Topics ranging from police activities, crime actions, government policies,disaster management,etc., are few of the clusters generated.

LDA does a good job of differentiating between topics 3 and 6. It should be noted that even though the words are similar, LDA differentiates based on the subjects mentioned in the headlines. But, there is a lot of un-related words in each topic. This implies that LDA fails to accurately cluster words into specific topics. This can be attributed to the fact that LDA uses conditional probabilities to make these clusters. The words appearing in multiple topics are the most frequent words in the headlines. Thus, LDA gives an idea of the words present in headlines related to various topics but fails with respect to uniqueness of words in the topics.

5.2 Non-negative Matrix Factorization

Non-negative matrix factorization (NMF or NNMF), also non-negative matrix approximation is a group of algorithms where a matrix is factorized into two matrices with the condition that all three consist of non-negative elements. This results in easier interpretability of the results. This can be used to cluster documents and model topics.

Given the original matrix A, two matrices W and H can be obtained, such that A= WH. NMF has an inherent clustering property, such that W and H represent the following information about A:

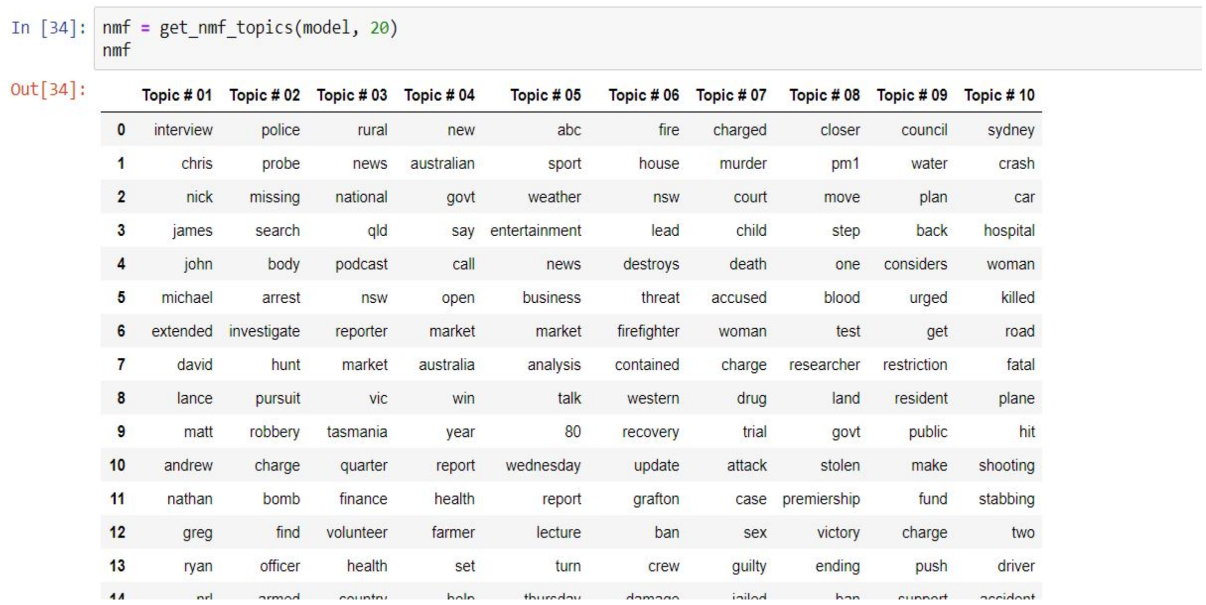
- A (Document-word matrix)—corpus that contains which words appear in which documents.
- W (Basis vectors)—the topics (headlines) discovered from the documents.
- H (Coefficient matrix)—the membership weights for the topics in each document. By using an objective function and optimizing it, W and H are calculated and updated by iterating them until both converge.

$$\frac{1}{2}\,||\mathbf{A}-\mathbf{WH}||_{\mathsf{F}}^{2}=\sum_{i=1}^{n}\sum_{j=1}^{m}(A_{ij}-(WH)_{ij})^{2}$$

Above is the objective function that is used in this model. Here the error of reconstruction between A and the product of W and H is minimized. Any distance metric like Euclidian, Manhattan, Jaccard, etc., can be used. The metric used here is Euclidian Distance. The update rules are derived from the function such that the error is minimized. The update rules obtained are:

$$W_{ic}\leftarrow W_{ic}\frac{(\mathbf{A}H)_{ic}}{(\mathbf{WH}H)_{ic}} \qquad H_{cj}\leftarrow H_{cj}\frac{(W\mathbf{A})_{cj}}{(W\mathbf{WH})_{cj}}$$

By updating these values simultaneously like Gradient Descent, the reconstruction error is recalculated. Repeating this process will result in convergence and minimum error.
For NMF, a design matrix is obtained. TF-IDF transformation is applied to the counts to improve results. A counts design matrix is obtained using Scikit's CountVectorizer module. This returns a matrix of size WxH (Headlines x Features) where each cell will contain the number of times a feature (Word) appears in that document(headline). For computational efficiency, only the 5000 best features are considered.
The counts are then transformed using the above mentioned TF-IDF transformer. These values are normalized such that length of each row is one (L1). To generate NMF, the most important scoring words in each cluster are obtained by iterating over each topic. The model is then executed to get the best topics in each cluster of the document.



```
In [34]: nmf = get_nmf_topics(model, 20)
         nmf
```

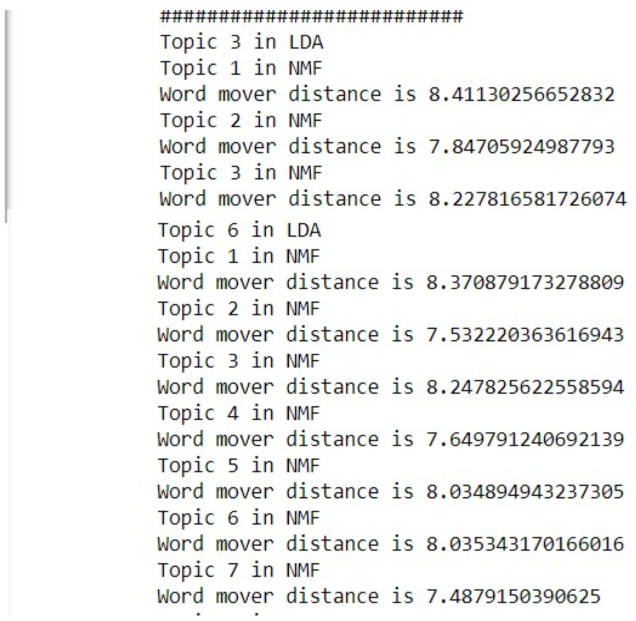
Out[34]:

	Topic # 01	Topic # 02	Topic # 03	Topic # 04	Topic # 05	Topic # 06	Topic # 07	Topic # 08	Topic # 09	Topic # 10
0	interview	police	rural	new	abc	fire	charged	closer	council	sydney
1	chris	probe	news	australian	sport	house	murder	pm1	water	crash
2	nick	missing	national	govt	weather	nsw	court	move	plan	car
3	james	search	qld	say	entertainment	lead	child	step	back	hospital
4	john	body	podcast	call	news	destroys	death	one	considers	woman
5	michael	arrest	nsw	open	business	threat	accused	blood	urged	killed
6	extended	investigate	reporter	market	market	firefighter	woman	test	get	road
7	david	hunt	market	australia	analysis	contained	charge	researcher	restriction	fatal
8	lance	pursuit	vic	win	talk	western	drug	land	resident	plane
9	matt	robbery	tasmania	year	80	recovery	trial	govt	public	hit
10	andrew	charge	quarter	report	wednesday	update	attack	stolen	make	shooting
11	nathan	bomb	finance	health	report	grafton	case	premiership	fund	stabbing
12	greg	find	volunteer	farmer	lecture	ban	sex	victory	charge	two
13	ryan	officer	health	set	turn	crew	guilty	ending	push	driver
14	nrl	armed	country	help	thursday	damage	jailed	ban	support	accident

After all the pre-processing steps were done, TF-IDF transformation was performed on the counts, the model was built and the headlines were fitted to the model. The above picture is a snapshot of the results of the model. At first glance, it can be seen that NMF performs well on the corpus. In topic 1, interview was clustered along with first names of people. This is expected since the headlines would be of the form ”In an interview with John”. Also, NMF differentiates between police activities and criminal activities accurately. Charge, search, investigate,pursuit, etc., are clustered into topic 2 while charged, murdered, accused, stabbing, etc., are clustered into topic 7. There are only a few words repeating in multiple topics and each topic mostly has words related to each other. This is because, NMF updates the topics and words simultaneously such that the two matrices together resemble the original headlines. This gradient descent like approach ensures that the final topics and words will closely resemble the headlines such that the error is minimum.

5.3 NMF vs LDA

It is clear that NMF performs better than LDA since there is less redundancy in the topics and the topics are well clustered when compared to the results of LDA. To look at the similarity of the two techniques, Word Mover’s Distance is calculated between the words of each topic generated by LDA and NMF.



```
##########################
Topic 3 in LDA
Topic 1 in NMF
Word mover distance is 8.41130256652832
Topic 2 in NMF
Word mover distance is 7.84705924987793
Topic 3 in NMF
Word mover distance is 8.22781658172607
Topic 6 in LDA
Topic 1 in NMF
Word mover distance is 8.370879173278809
Topic 2 in NMF
Word mover distance is 7.532220363616943
Topic 3 in NMF
Word mover distance is 8.247825622558594
Topic 4 in NMF
Word mover distance is 7.649791240692139
Topic 5 in NMF
Word mover distance is 8.034894943237305
Topic 6 in NMF
Word mover distance is 8.035343170166016
Topic 7 in NMF
Word mover distance is 7.4879150390625
```

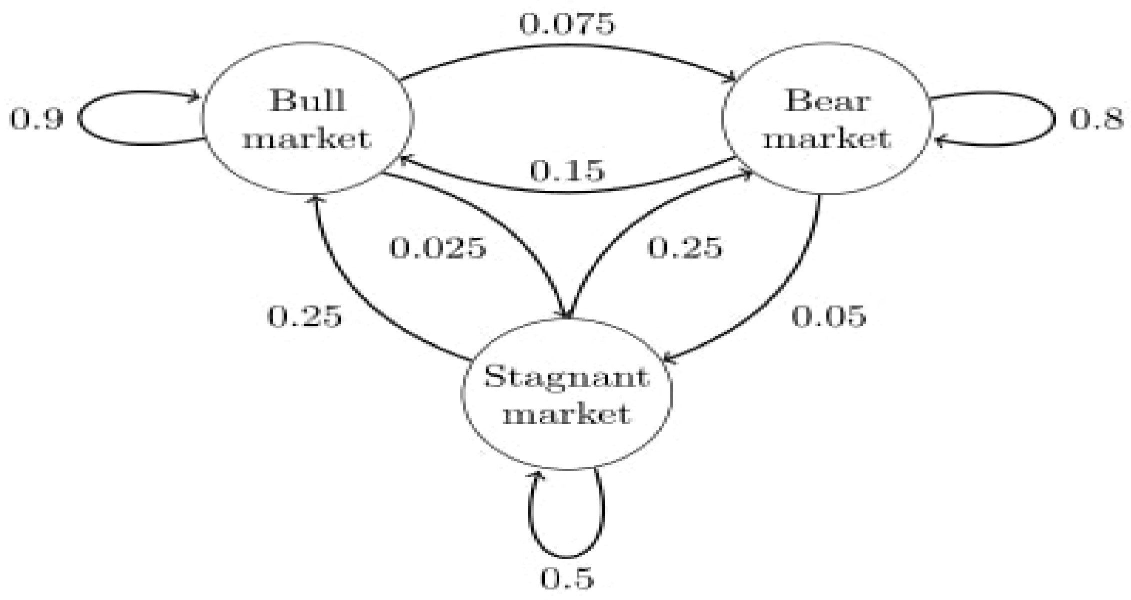
Topic 3 of LDA is closest to topic 2 of NMF. Incidentally, these are the topics related to police activities. Likewise, topic 6 of LDA is closest to topic 7 of NMF and these are the criminal activity related clusters in the two models. Also, topic 4 of NMF has low distance with all of the topics of LDA. This is because, topic 4 has a lot of trivial and common words which are present in almost all the topics of LDA. To conclude this part, NMF performs much better than LDA and the topics generated in both have a degree of similarity between them.

5.4 Markov Chains

In the above section, we have talked about Natural Language Processing using LDA, NFM for topic modelling and in this section, we will talk about Natural Language Generation, which is considered as the opposite of Natural Language Processing. Natural Language Generation(NLG) creates sentences based on existing corpora for text. This method is extremely useful in multiple applications like automatically creating textual reports from dashboards, automated reply to customer queries, providing seed content for writers to develop upon to name a few.

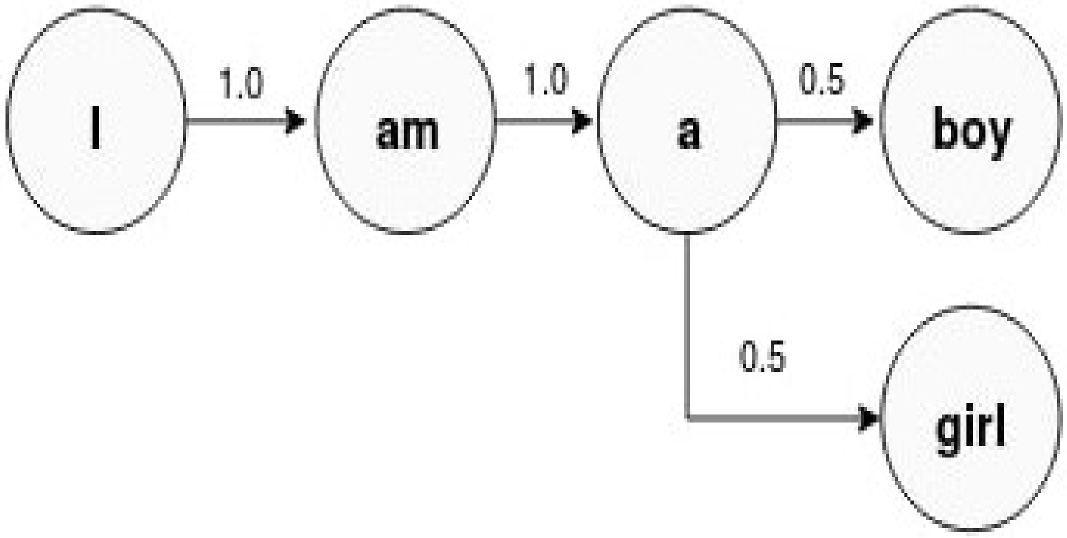
The simplest way we can build a NLG model is to use to use the existing corpora and modify some common words, for example, the word color could be replaced by "red" or "blue", similarly, we can build a simple mapping file containing alternate words for common words to generate new sentences. This method is extremely inefficient because we will need to manually create the mapping file, which needs to be updated from time to time and there would be a huge overlap with the original text. So, to address these issue Markovify could be used which uses Markov chain as the backed.

Before we get into the working of the Markovify package lets first understand how Markov chain works. The Markov model helps us understand a sequence of events using probability by looking at how these events have occurred in the past and in what order.

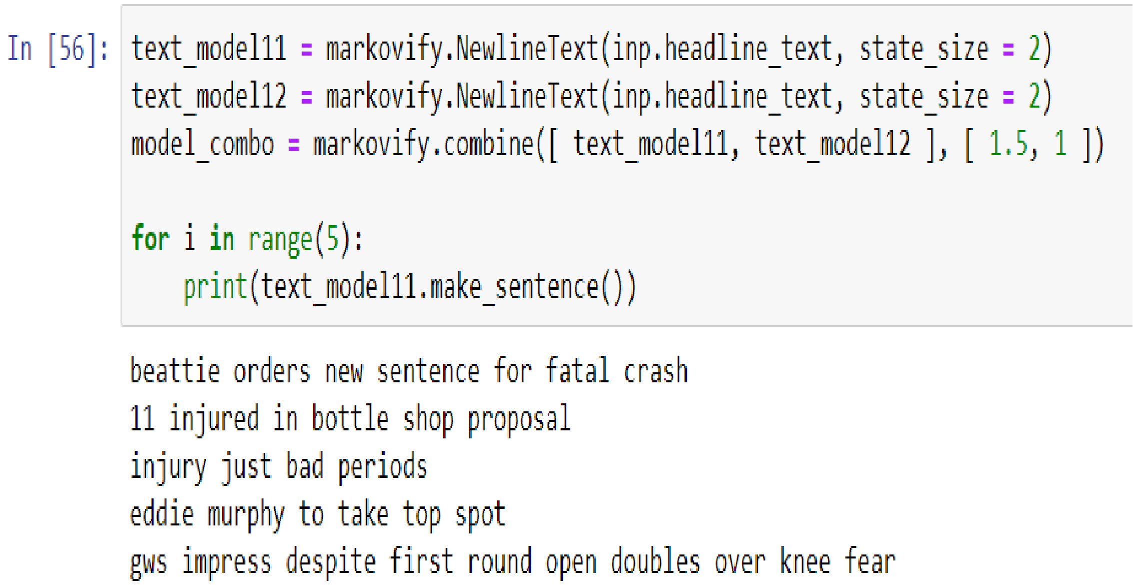


In general Markov chains are represented by a state diagram, the above image has three states one “Bull Market”, “Bear Market” and the third one being “Stagnant Market”. The values represent the transition probability between the states. Using this we can better understand how the market works and plan strategies accordingly.

In the case of text generation, the model takes an initial set of words selected randomly and identifies the most likely word that could follow it based on the corpus that the model was trained upon. One important parameter for the Markovify is the number of states, a state size of two means that the model looks at one word as the initial word and identifies the next best word, then it uses the word generated to identify the word next to it and this keeps repeating. The model with a state size of 3 is very similar to two state model except it generates a new word based on two prior words instead of one. Model with two states performs well in generating sentences but sometimes the sentences are heavily influenced by the corpora and could result in a lot of random sentences. The three state model generally creates more meaningful sentences but the model fails to generate sentences sometimes due to low probability.



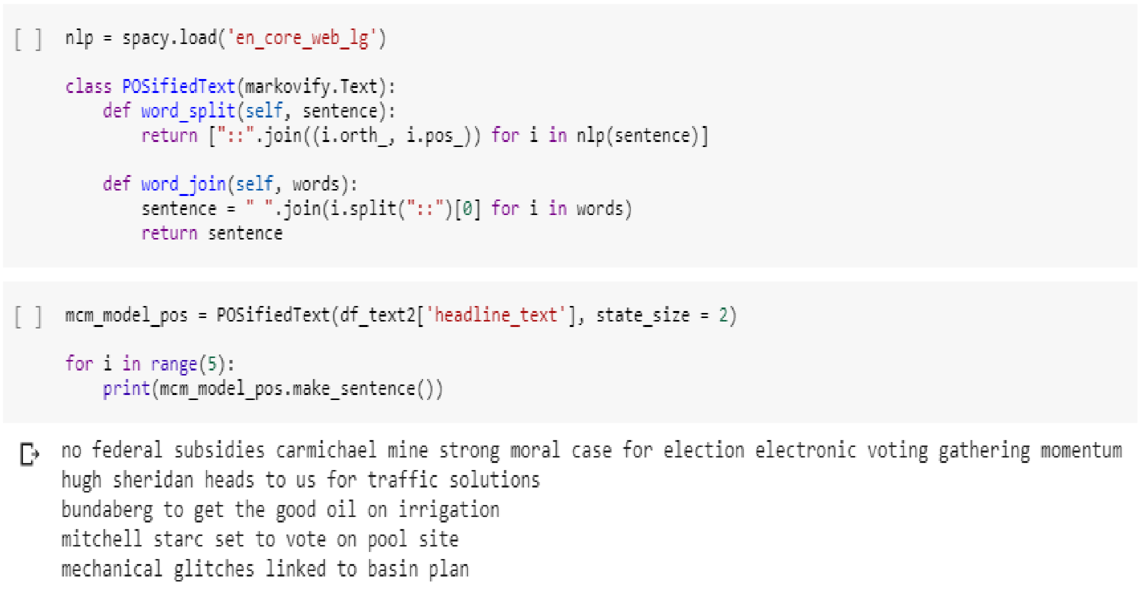
One way the model makes sure that it does not generate pre-existing sentences is by looking at the similarity of the generated sentence to original corpora. If more than seventy percent of the sentences are similar, then the generated sentence is rejected and another sentence is generated. A basic Markov chain model is able to generate good sentences but if needed, it could be improved by using the following methods. The first method is using an ensemble of models created by varying the state parameter. The second method is to by using POS tagging using the spaCy package, this could greatly improve the quality of sentences.



```
In [56]: text_model11 = markovify.NewlineText(inp.headline_text, state_size = 2)
         text_model12 = markovify.NewlineText(inp.headline_text, state_size = 2)
         model_combo = markovify.combine([ text_model11, text_model12 ], [ 1.5, 1 ])

         for i in range(5):
             print(text_model11.make_sentence())
```

```
beattie orders new sentence for fatal crash
11 injured in bottle shop proposal
injury just bad periods
eddie murphy to take top spot
gws impress despite first round open doubles over knee fear
```

The above results show how the random sentences are generated from combining two different Markov models, please note that the state size has to be the same else the models could not be combined together. As stated above, the results could be improved by using POS tags, this helps in eliminating the sentences that do not follow the general grammatical structure. The spacy package was used to add the POS information. Below are the results after adding the POS tags.



```
[ ] nlp = spacy.load('en_core_web_lg')

    class POSifiedText(markovify.Text):
        def word_split(self, sentence):
            return ["::".join((i.orth_, i.pos_)) for i in nlp(sentence)]

        def word_join(self, words):
            sentence = " ".join(i.split("::")[0] for i in words)
            return sentence
```

```
[ ] mcm_model_pos = POSifiedText(df_text2['headline_text'], state_size = 2)

    for i in range(5):
        print(mcm_model_pos.make_sentence())
```

```
no federal subsidies carmichael mine strong moral case for election electronic voting gathering momentum
hugh sheridan heads to us for traffic solutions
bundaberg to get the good oil on irrigation
mitchell starc set to vote on pool site
mechanical glitches linked to basin plan
```

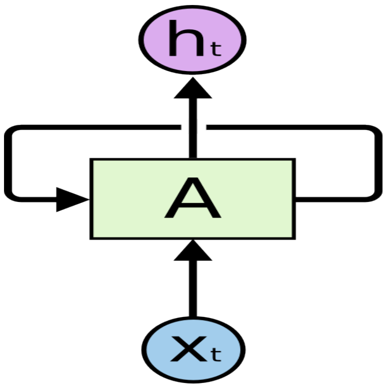
One problem with working with a large corpus is that the model runs into memory issues as it retains the original text. This could be resolved by specifying not to store the original corpora. The package uses punctuation marks to identify the start and end of a sentence, so it is advised to get a well-punctuated corpus for training purposes.

Markovify performs quite well in generating meaningful results but the sentence generation process could be improved using LSTM.

5.5 Long Short Term Memory

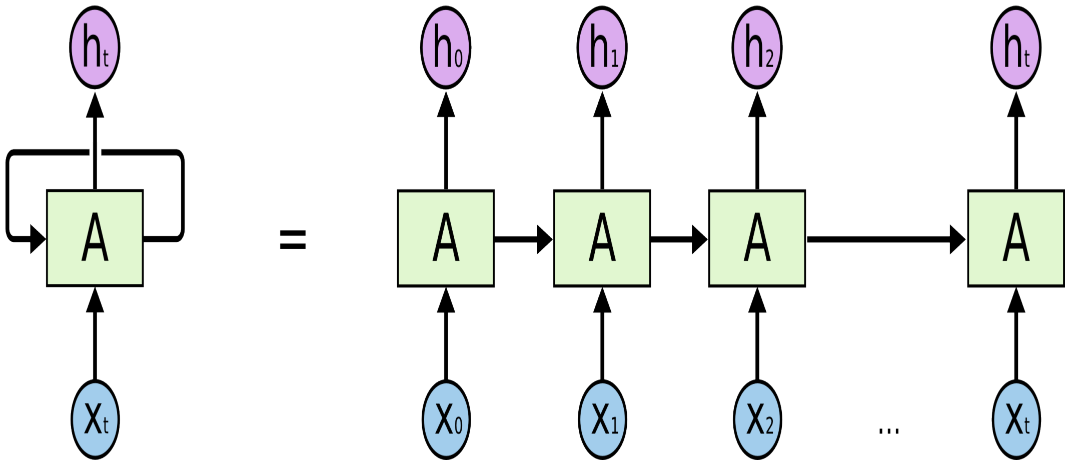
Language modeling is the problem in Natural Language Processing involving speech to text, chat bot and text summarization. Text generation is one such language modelling problem. It basically calculates the likelihood of occurrence of a word based on the previous words used in the particular sentence (or) text.
In this project, a recurrent neural network composed of Long Short-term memory units would be implemented to generate headlines based on a given input.

Working of Recurrent Neural Networks (RNN) :
If we would like to classify what kind of action/event is happening at every point in a movie (or) video, traditional neural networks do not work since they do not have any information about the previous events. A recurrent neural network will be helpful in this scenario since they have loops present in them allowing information to be saved.



The state A receives an input x_t and outputs a value h_t. The loop present allows the obtained information to be passed on to the next state in the network.

An expansion of the above network can be represented as:



Eventually a RNN boils down to multiple copies of the same network with each passing a particular message/information to its successor.

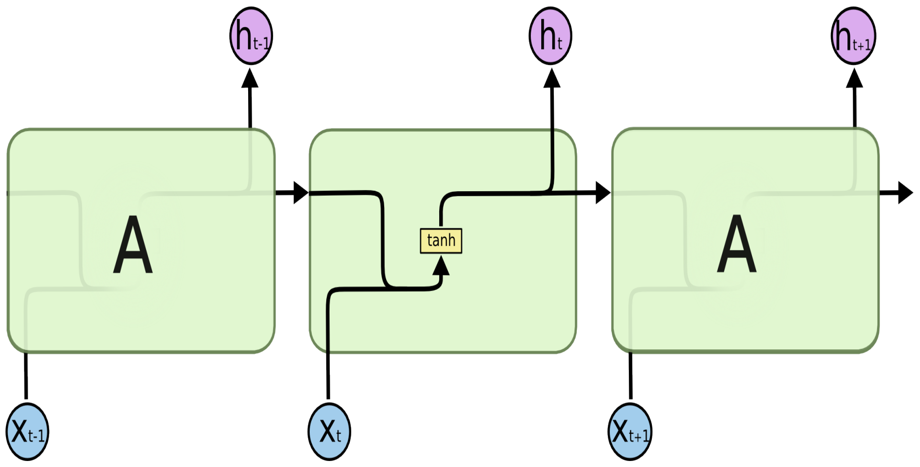
Working of LSTM :
A traditional RNN can save and process information from one state to another but this can happen only if the gap between the information and the place where it is needed is small.

For example, if we are to predict the last word in the sentence “the player used Nike boots to score goal”, in this scenario it is very evident that the last word is “goal” and we do not need a lot of information from the past to predict the word. For such cases, a small RNN with few states would be sufficient to predict the last word in the sentence.

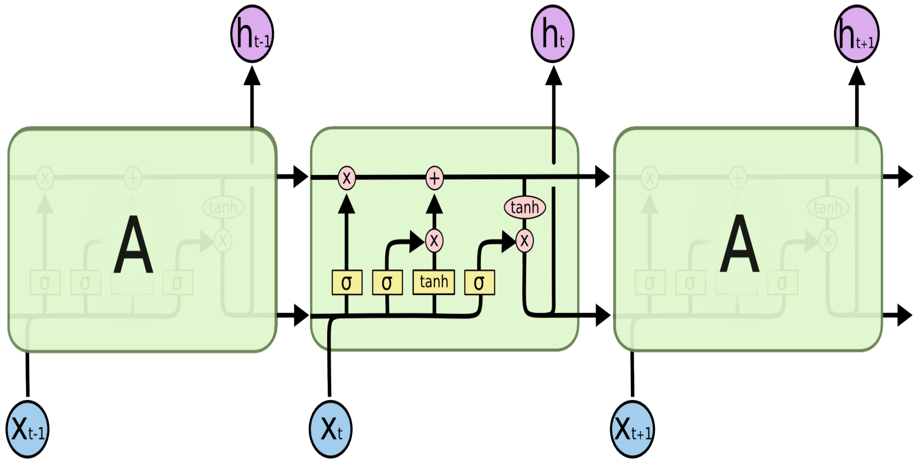
However, if we are to predict the last word in a sentence such as: “The player grew up in Spain and played there for 10 years which why he speaks Spanish”. In this case, the most recent information would help us know that the last word is a language, but we need the context of the country – “Spain” to narrow down on the language. In situations where the gap between the relevant information and the place where it is needed is large, it requires a lot of human effort to pick the parameters of RNN very carefully to enable it to capture the long-term dependencies and it might be difficult to do this in all scenarios.

This eventually led to the Long Short-term Memory networks which are a special kind of RNN capable of learning long-term dependencies. They are designed to remember information for a longer period.

All RNNs will have a chain of repeating modules of Neural network and in a standard RNN, there will be a single tanh layer like the one shown below:



LSTMs also have similar structure but the repeating modules are different. Instead of a single tanh layer, they have four interacting layers.



The horizontal line at the top of the cell state acts as a conveyor belt processing the entire information with few minor linear interactions (”+” and ”x”).

The sigmoid functions allows the cell states to remove or drop information since it’s output ranges from 0 to 1 where a value of “0” does not allow any information to pass through the cell state.

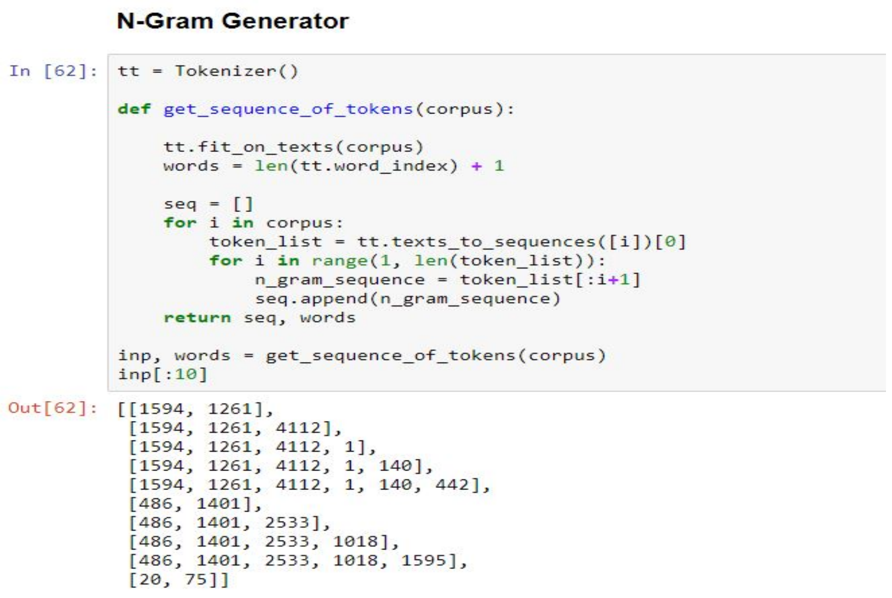
Architecture used for this project:

1. Input Layer: Sequences of words are given as input to the model
2. LSTM Layer: The output is obtained through the LSTM units. 100 units are being added in the layer
3. Droput Layer: This layer helps in preventing overfitting by turning off the activations of some of the neurons randomly in the LSTM layer
4. Ouptut Layer: Computes the probability of the next possible word as the output

The model will be run for 50 epochs.

Model Building Methodology :

1. Tokenization to create sequence of N-gram tokens:

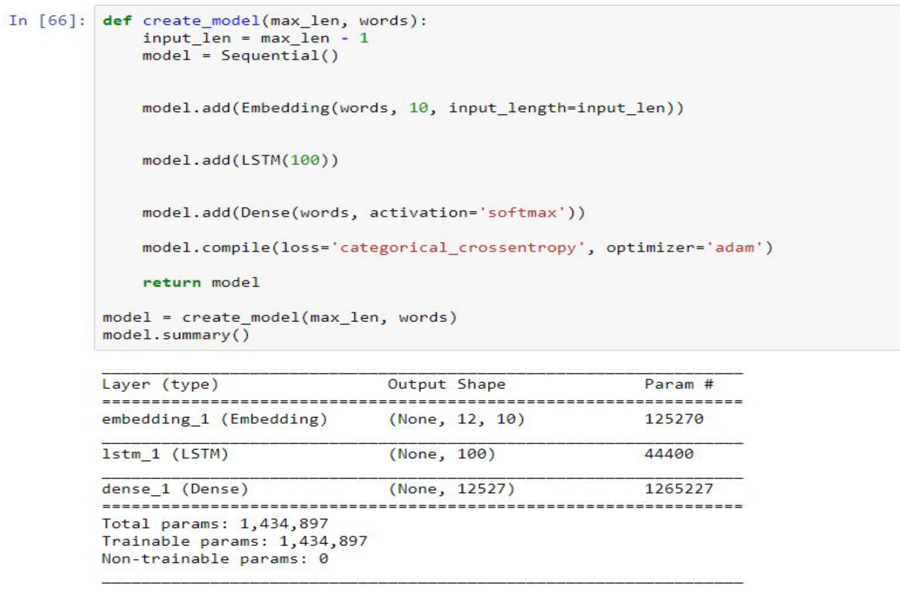


N-Gram Generator

```
In [62]: tt = Tokenizer()

         def get_sequence_of_tokens(corpus):

             tt.fit_on_texts(corpus)
             words = len(tt.word_index) + 1

             seq = []
             for i in corpus:
                 token_list = tt.texts_to_sequences([i])[0]
                 for i in range(1, len(token_list)):
                     n_gram_sequence = token_list[:i+1]
                     seq.append(n_gram_sequence)
             return seq, words

         inp, words = get_sequence_of_tokens(corpus)
         inp[:10]

Out[62]: [[1594, 1261],
          [1594, 1261, 4112],
          [1594, 1261, 4112, 1],
          [1594, 1261, 4112, 1, 140],
          [1594, 1261, 4112, 1, 140, 442],
          [486, 1401],
          [486, 1401, 2533],
          [486, 1401, 2533, 1018],
          [486, 1401, 2533, 1018, 1595],
          [20, 75]]
```

The above output indicates the n-gram tokens/phrases generated from the input corpus with each number indicating the index of the word in the corpus.

2. Model Architecture :

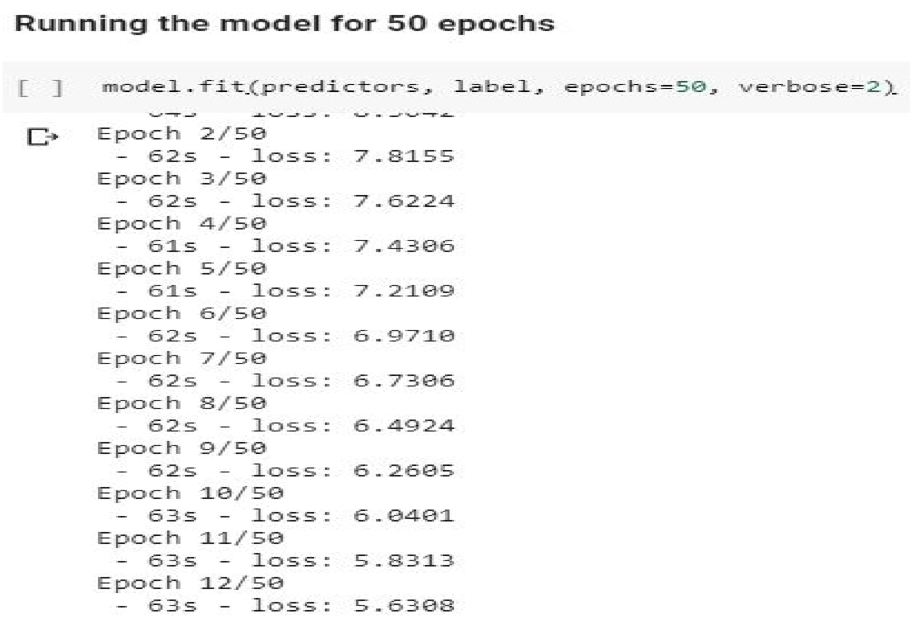


```
In [66]: def create_model(max_len, words):
             input_len = max_len - 1
             model = Sequential()


             model.add(Embedding(words, 10, input_length=input_len))


             model.add(LSTM(100))


             model.add(Dense(words, activation='softmax'))

             model.compile(loss='categorical_crossentropy', optimizer='adam')

             return model

         model = create_model(max_len, words)
         model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 12, 10)            125270
_________________________________________________________________
lstm_1 (LSTM)                (None, 100)               44400
_________________________________________________________________
dense_1 (Dense)              (None, 12527)             1265227
=================================================================
Total params: 1,434,897
Trainable params: 1,434,897
Non-trainable params: 0
_________________________________________________________________
```

The model is executed with an input layer containing the sequence of words followed by the LSTM layer consisting of 100 units and an output layer consisting of softmax function with Adam's optimizer. The model is evaluated on crossentropy.

3. Execution of the model :



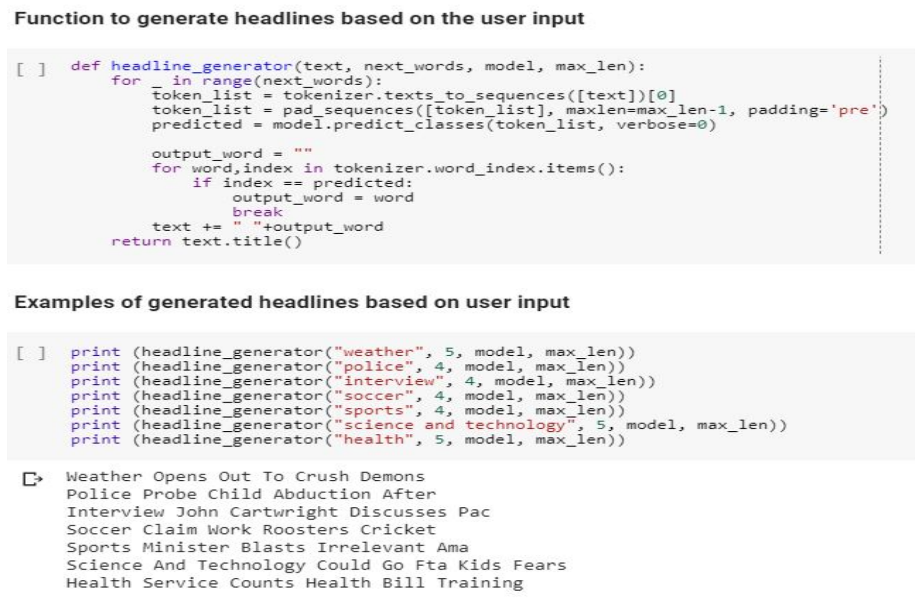
Running the model for 50 epochs

```
[ ]  model.fit(predictors, label, epochs=50, verbose=2)
```

```
Epoch 2/50
 - 62s - loss: 7.8155
Epoch 3/50
 - 62s - loss: 7.6224
Epoch 4/50
 - 61s - loss: 7.4306
Epoch 5/50
 - 61s - loss: 7.2109
Epoch 6/50
 - 62s - loss: 6.9710
Epoch 7/50
 - 62s - loss: 6.7306
Epoch 8/50
 - 62s - loss: 6.4924
Epoch 9/50
 - 62s - loss: 6.2605
Epoch 10/50
 - 63s - loss: 6.0401
Epoch 11/50
 - 63s - loss: 5.8313
Epoch 12/50
 - 63s - loss: 5.6308
```

The above image is a snapshot of the execution of the LSTM model for 50 epochs.

4. Generating Sentences based on User Input :
   The model predicts a sequence of words that follow a given input and these words are padded together to form meaningful headlines.



Function to generate headlines based on the user input

```
[ ]  def headline_generator(text, next_words, model, max_len):
         for _ in range(next_words):
             token_list = tokenizer.texts_to_sequences([text])[0]
             token_list = pad_sequences([token_list], maxlen=max_len-1, padding='pre')
             predicted = model.predict_classes(token_list, verbose=0)

             output_word = ""
             for word,index in tokenizer.word_index.items():
                 if index == predicted:
                     output_word = word
                     break
             text += " "+output_word
         return text.title()
```

Examples of generated headlines based on user input

```
[ ]  print (headline_generator("weather", 5, model, max_len))
     print (headline_generator("police", 4, model, max_len))
     print (headline_generator("interview", 4, model, max_len))
     print (headline_generator("soccer", 4, model, max_len))
     print (headline_generator("sports", 4, model, max_len))
     print (headline_generator("science and technology", 5, model, max_len))
     print (headline_generator("health", 5, model, max_len))
```

```
Weather Opens Out To Crush Demons
Police Probe Child Abduction After
Interview John Cartwright Discusses Pac
Soccer Claim Work Roosters Cricket
Sports Minister Blasts Irrelevant Ama
Science And Technology Could Go Fta Kids Fears
Health Service Counts Health Bill Training
```

The above output indicates some of the headlines that are generated based on the input phrases provided by the user.

6 Conclusion

Through this project we have successfully formulated a framework to generate topics from a given corpus through LDA and NMF models. It was observed that NMF performs better than LDA since there is less redundancy in the topics and the topics are well clustered when compared to LDA. The generated cluster of words will help the users understand the overview of topics in the corpus. Also, results from Morkov chains model provides the user with more meaningful phrases from the corpus with a more human-like topics. These gives the user an idea to also provide an input to retrieve input-specific summaries obtained from LSTM technique.

This can be extended in developing an autonomous system that generates topics based on a provided corpus and further generate meaningful headlines upon user's input and previously chosen topics.

7 References

[1] https://medium.com/ml2vec/topic-modeling-is-an-unsupervised-learning-approach-to-clustering-documents-to-discover-topics-fdfbf30e27df
[2] https://www.kaggle.com/nulldata/meaningful-random-headlines-by-markov-chain/notebook
[3] https://www.kaggle.com/shivamb/beginners-guide-to-text-generation-using-lstms/data
[4] http://colah.github.io/posts/2015-08-Understanding-LSTMs/
[5] https://github.com/jsvine/markovify
[6] https://medium.com/@rahulvaish/textblob-and-sentiment-analysis-python-a687e9fabe96