Text Summarization

Summarize the news to the world!

University of Science and Technology at Zewail City
CIE 553 - Natural Language Processing course

Team Information

- Asmaa Ibrahim 201701056
- Elsayed Mostafa 201700316
- Salma Elbess 201601152
- Muhammed Alasmar 201700038

Motivation

In this fast world, text summarization is an important task to create small digestible information content instead of reading the whole text.

Applications of text summarization:

- Chat bots history tracking - Media monitoring - Meetings summary
- Book summaries - .. etc

Why news summarization ?

Summarization of news will allow more people the opportunity to be more aware of the events all over the world without the need to spend lots of time on reading a whole article.

Automation of that process will make more pieces of summaries available instantly.

Objectives

- Generating a headline for the text.
- Using **the extractive approach** at first as a traditional NLP approach.
- Using deep learning models for **abstractive approach**.
- Using appropriate similarity measurement such as **ROUGE** metric.

Summarization Approaches

Extractive Summarization

Choosing whole sentences from the original text that score highly in order to represent the text.

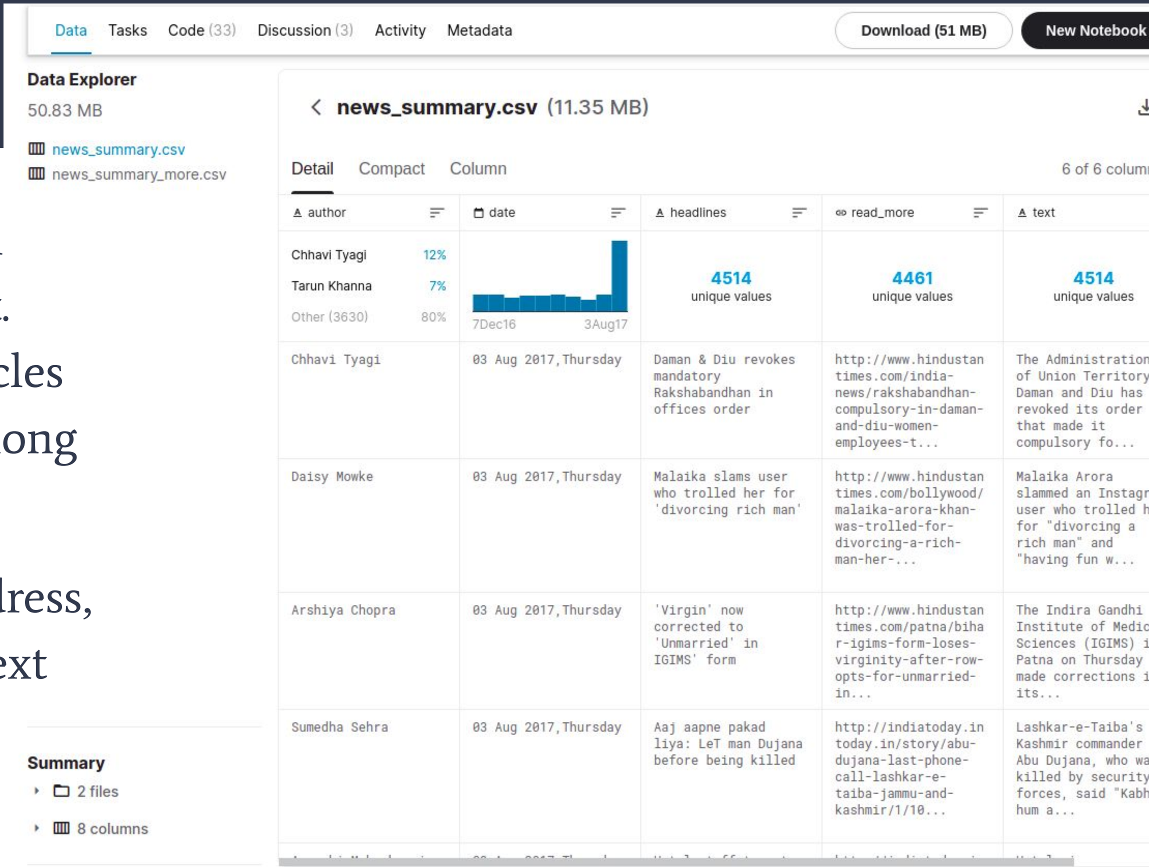
Abstractive Summarization

A more advanced approach that aims at choosing the important sections in order to create context and generate new sentences.

We did BOTH !

Dataset

- News Summary Dataset found on [Kaggle](#) was chosen for the project.
- The dataset has 4515 of news articles scrapped from inshorts website along with their summaries.
- Columns; Author, Date, Web Address, Headline, Summary, Complete Text



Extractive

Summarization

Extractive Summarization

- How to identify the sentence importance ?
  - TF-IDF algorithm
  - TextRank algorithm

TF-IDF based summarization

- Two types of scores:
    - **Word score :** the frequency of the word in a specific sentence weighted by its frequency along the whole sentences.
    - **Sentence score :** the summation of its words' scores.

TF-IDF Implementation

- This algorithm is implemented from scratch in python.

Main Functions

```
def apply_preprocessing(text):
    original_sentences = sent_tokenize(text.lower())
    sentences = []
    # Sentences Pre-processing
    for sent in original_sentences:
        tokens = [item for item in word_tokenize(sent) if item not in my_stopwords and item != '.' and item.isalpha()] # Pr
eprocessing
        sentences.append(' '.join([stemmer.stem(token) for token in tokens])) # Stemming
    return original_sentences, sentences

def word_tf(word,sentence):
    #tf_score =  sentence.count(word)/ len(sentence)
    return sentence.count(word)/ len(sentence)

def word_idf(word, sentences):
    return math.log10(' '.join(sentences).count(word)/ len(sentences))

def word_in_sentence_tfidf(word, sentence, sentences):
    return word_tf(word,sentence)*word_idf(word, sentences)

def sentence_score(sentence, sentences):
    return sum([word_in_sentence_tfidf(word, sentence, sentences) for word in sentence])

def sentences_scores(sentences):
    return [sentence_score(sent, sentences) for sent in sentences]

def get_best_k_sentences_indecies(k, sentences, original_sentences):
    #lista = sorted(list(np.argsort(sentences_scores(sentences))[-k:-1]))
    return sorted(list(np.argsort(sentences_scores(sentences))[-k:]))

def get_best_k_sentences(k, sentences, original_sentences):
    return [original_sentences[idx] for idx in get_best_k_sentences_indecies(k, sentences, original_sentences)]

def summarize(text, k):
    # Apply preprocessing
    original_sentences, sentences = apply_preprocessing(text)
    # Summariez
    return ' '.join(get_best_k_sentences(k, sentences, original_sentences))
```

TF-IDF Implementation

The news:

Indian Super League side Atletico de Kolkata has been renamed to 'Aamar, Tomar Kolkata' (ATK) after the team's partnership with Spanish football club Atletico Madrid ended. The Spanish club, which had provided technical support to the Indian club, will be selling their stake in the club to the club's principal owner Sanjiv Goenka.

The generated summary:

```
# Take a look at the
ext_summaries[0]
```

```
"indian super league side atletico de kolkata has been renamed to 'aamar, tomar kolkata' (atk) after the team's partnership
with spanish football club atletico madrid ended."
```

Comparison with BERTSUM

- BERTSUM is a version of the famous BERT model. It’s a pre-trained transformed model used for extractive summarization.

- In general, BERT models achieved record-breaking performance on multiple NLP tasks.

- Using ROUGE-1 metric, to average the precision and recall over the sentences. We got acceptable accuracy.

```
{'f': 0.7357794890362269, 'p': 0.8368993562845303, 'r': 0.6861554970380593}
```

Comparison with BERTSUM

- However, the exact recall, precision are really higher! Why?
  Here are examples of the non-matched summaries !

```
BERT:  prime minister narendra modi on monday accompanied his visiting australian counterpart malcolm turnbull on a metro ri
de in new delhi. the two prime ministers boarded the metro at the mandi house station and headed towards the akshardham temp
le. "
OURS:  prime minister narendra modi on monday accompanied his visiting australian counterpart malcolm turnbull on a metro ri
de in new delhi.
=========
BERT:  actor randeep hooda visited the kargil war memorial in dras ahead of kargil vijay diwas, which is observed on july 2
6. "
OURS:  actor randeep hooda visited the kargil war memorial in dras ahead of kargil vijay diwas, which is observed on july 2
6.
=========
BERT:  the two-day unesco natural heritage festival began on saturday at great himalayan national park in sai ropa, himachal
pradesh. the wildlife institute of india said the festival will feature a media workshop on environmental journalism, discus
sions on natural and cultural heritage of the himalayan region and a heritage walk.
OURS:  the wildlife institute of india said the festival will feature a media workshop on environmental journalism, discussi
ons on natural and cultural heritage of the himalayan region and a heritage walk.
=========
```

The difference is due to different pre-processing applied or missing at BERT!

Comparison with BERTSUM

- Efficiency-wise:
  - TF-IDF approach

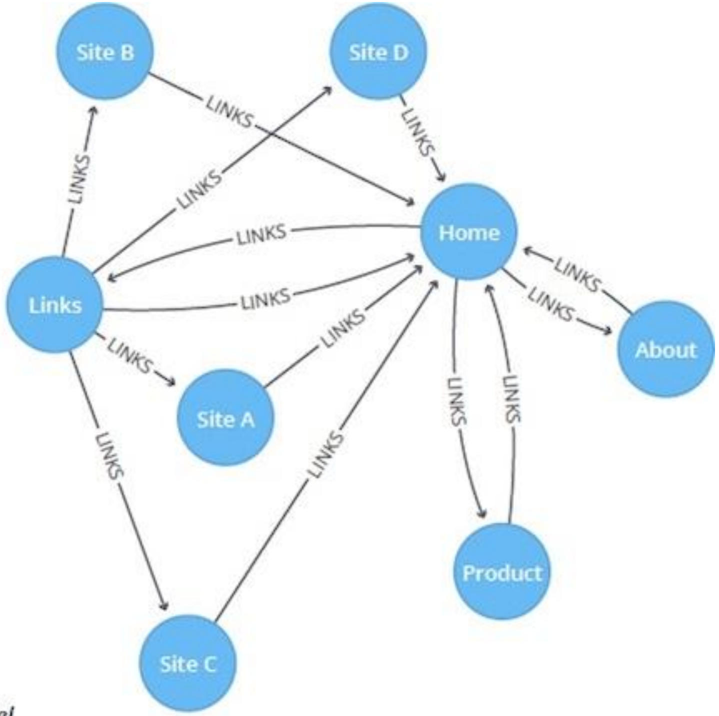
```
Elapsed time: 21.258 seconds for 3611 sentences
```

  - BERT model

```
Elapsed time: 3.3252482414245605 seconds FOR 2 sentences!!
```

TextRank based summarization

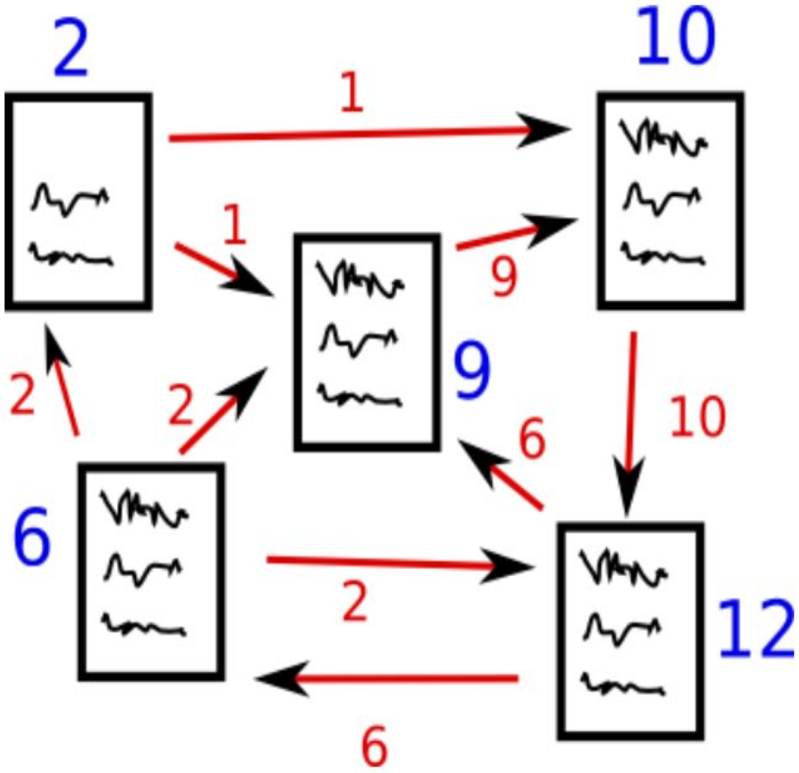
- Google's PageRank Algorithm
- How each page's rank is computed ?
- Similarity Matrix formation



Graph Model

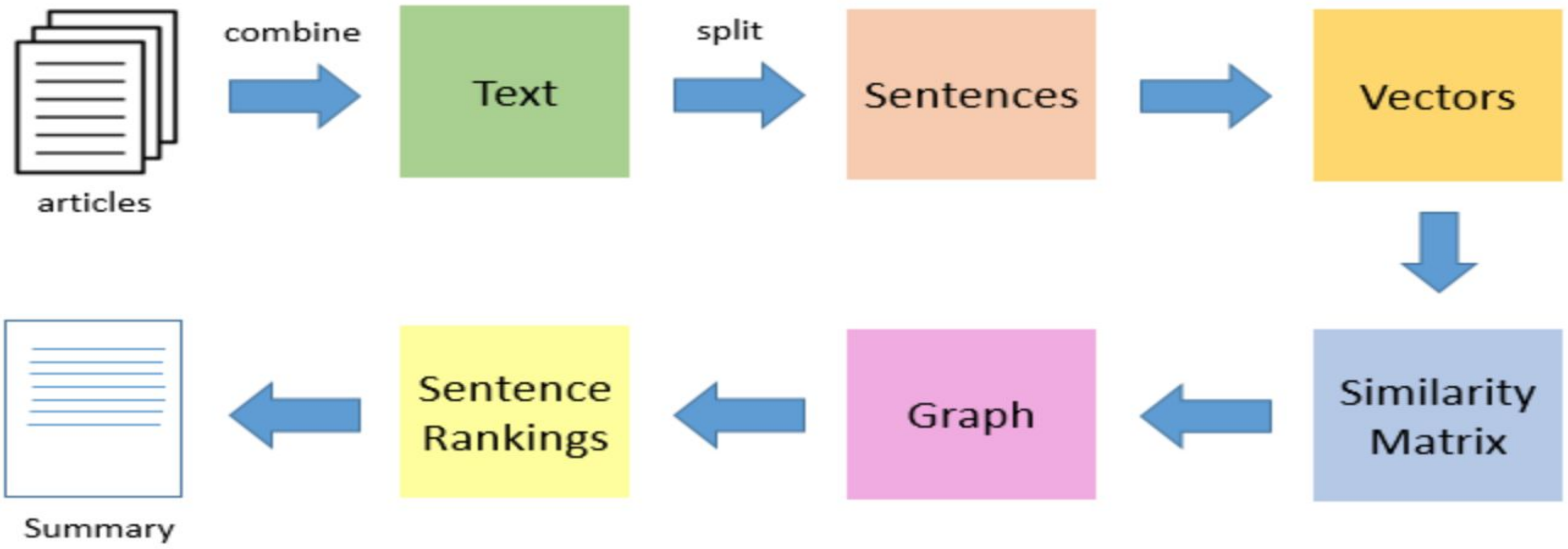
Continued;

- How TextRank is related to PageRank ?
- Using Cosine similarity to compute the similarity matrix.
- Initializing sentences' ranks by 1 at first.
- Then updating each sentence rank by summing over other sentences' updated ranks weighted by their similarity from similarity matrix.



Wrap up

TextRank(Sentence1) = (1 - C) + C*(similarity[sentence 1,sentence 2]*TextRank(Sentence 2)



Comparison with TF-IDF

The news:

Tamil Nadu Milk and Dairy Products Development Minister Rajenthra Bhalaji has alleged that products of private milk producers are adulterated. In a press briefing, Bhalaji held milk products by Nestle and Reliance, affirming that he had laboratory results which show that these are contaminated. He further alleged that there are contents of caustic soda and bleaching powder in the products.

The generated summary:

- *Using TF-IDF:*

```
===========================================
the generated summary:  tamil nadu milk and dairy products development minister rajenthra bhalaji has alleged that products of private milk producers are adulterated.
```

- *Using TextRank:*

```
===========================================
the generated summary:  tamil nadu milk and dairy products development minister rajenthra bhalaji has alleged that products of private milk producers are adulterated.
```

Comparison with TF-IDF

- *<u>Efficiency-wise:</u>*
    - TF-IDF approach

```
Elapsed time: 21.258 seconds for 3611 sentences
```

    - TextRank approach

```
  app.launch_new_instance()
Elapsed time is 314.064 seconds
```

- *<u>Precision (Considering the TF-IDF as the base)=</u>* 73.6 %

Abstractive

Summarization

Abstractive Summarization

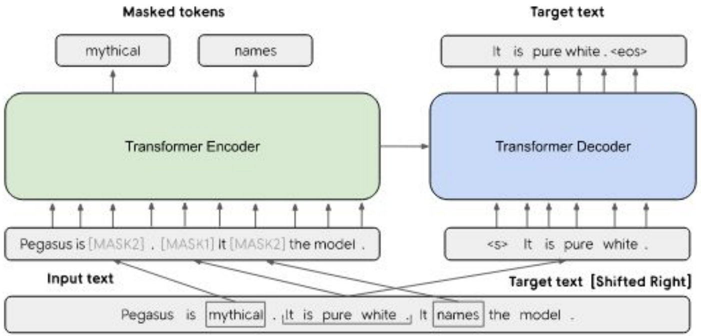
In order attempt toward an efficient abstractive summarization model.
We have decided to go along these two ways:

1. Use pre trained famous models that were proved to produce good summarization results
    a. Google's Pegasus
    b. Facebook's bart-large
    c. Roberta2Roberta

2. Implement an abstractive model from scratch, to get a hint of how it works and to be able to optimize the model

PEGASUS

- PEGASUS is a state-of-the-art abstractive model for nlp summarization.
- It was trained on different datasets. We used the one trained on XNUM
- The training of PEGASUS was done by trying gap sentences filling rather than direct summaries.



PEGASUS Results

Original Text:
The new Nokia 3310 has a 2.4-inch colour screen unlike the original phone, which came with an 84x84 black and white display. The 2017 Nokia 3310's 1200mAh battery life is 10 times more than the original. The new model, which is slimmer and lighter than the original, has an updated version of the 'Snake' game, which was made by Gameloft

Original headline:
How is the new Nokia 3310 different from its older version?

Generated headline:
Nokia has released a new version of its 3310 mobile phone.

```
google/pegasus-xsum rouge-1 score
{'f': 0.15234493824976872, 'p': 0.23641774891774894, 'r': 0.12066067483714545}
```

Bart

- BART is a denoising autoencoder for pretraining sequence-to-sequence models.
- BART is trained by corrupting text with an arbitrary noising function, and learning a model to reconstruct the original text.

BART Results

Original Text:
The new Nokia 3310 has a 2.4-inch colour screen unlike the original phone, which came with an 84x84 black and white display. The 2017 Nokia 3310's 1200mAh battery life is 10 times more than the original. The new model, which is slimmer and lighter than the original, has an updated version of the 'Snake' game, which was made by Gameloft

Original headline:
How is the new Nokia 3310 different from its older version?

Generated headline:
The new Nokia 3310 has a 2.4-inch colour screen unlike the original

```
facebook/bart-large-cnn rouge-1 score
{'f': 0.3421554529896391, 'p': 0.38953463203463207, 'r': 0.31358477307006716}
```

Roberta2Roberta

- Roberta2Roberta was fined tuned for summarization

- Although it is not a state-of-the-art model, it produces reasonable summarization results

- It was also trained on another news summarization dataset, which is why we though it should yield good results.

Roberta2Roberta Results

Original Text:
The new Nokia 3310 has a 2.4-inch colour screen unlike the original phone, which came with an 84x84 black and white display. The 2017 Nokia 3310's 1200mAh battery life is 10 times more than the original. The new model, which is slimmer and lighter than the original, has an updated version of the 'Snake' game, which was made by Gameloft

Original headline:
How is the new Nokia 3310 different from its older version?

Generated headline:
Nokia has revealed that its flagship mobile phone is back on sale in the UK and Ireland

```
google/roberta2roberta_L-24_bbc rouge-1 score
{'f': 0.1400761432407765, 'p': 0.2193217893217893, 'r': 0.10463552823066216}
```

Preprocessing for SEQ2SEQ Abstractive Model

- Before our text corpus could be used to train a deep learning seq2seq model, we had to do some preprocessing.
  1. We used contraction mapping dictionary to substitute abbreviations (e.g. you’re: you are)
  2. We used lower cased sentences
  3. Removed redundant punctuations and left alphanumeric characters
  4. We’ve removed stop words from the text **not the headlines**

- **After those steps we had to tokenize the words:**
  1. Tokenization
  2. Text to sequences
  3. Padded sequences according to the maximum number of words in a sequence

```python
def preprocess_text(textset):
    '''
    This function applies necessary preprocessing on the text
    Inputs:
        sentences [array or list containing the sentences]
    Outputs:
        processed text
    '''
    stop_words = set(stopwords.words('english'))
    cleaned_text = []
    for text in textset:
        # Replacine Irregular Spaces with regular ones
        new_text=re.sub("(\\t)", ' ', text).lower()
        new_text=re.sub("(\\r)", ' ', new_text)
        new_text=re.sub("(\\n)", ' ', new_text)
        new_text=re.sub("(__+)", ' ', str(new_text)).lower()   #remove _ if it occors more than one time consecutively
        new_text=re.sub("(--+)", ' ', str(new_text)).lower()   #remove - if it occors more than one time consecutively
        new_text=re.sub("(~~+)", ' ', str(new_text)).lower()   #remove ~ if it occors more than one time consecutively
        new_text=re.sub("(\+\++)", ' ', str(new_text)).lower()   #remove + if it occors more than one time consecutively
        new_text=re.sub("(\.\.+)", ' ', str(new_text)).lower()   #remove . if it occors more than one time consecutively

        new_text=re.sub(r"[<>()|&©ø\[\]\'\",;?~*!]", ' ', str(new_text)).lower() #remove <>()|&©ø"',;?~*!

        # Contraction mapping
        new_text = ' '.join([contraction_mapping[word] if word in contraction_mapping.keys() else word for word in new_text.split()])
        # Removing 's
        new_text = re.sub(r"'s\b|s'\b","",new_text)
        # Removing any non alphanumeric character
        new_text = re.sub("[^a-zA-Z0-9\ ]", "", new_text)
        # Removing Stop words
        new_text = ' '.join([word for word in new_text.split() if word not in stop_words])

        cleaned_text.append(new_text)
    return cleaned_text
```

```python
def preprocess_summary(summaryset):
    '''
    This function applies necessary preprocessing on the summaries "headlines"
    Inputs:
        sentences [array or list containing the sentences]
    Outputs:
        processed summaries
    '''
    cleaned_summary = []
    for summary in summaryset:
        # Replacine Irregular Spaces with regular ones
        new_text=re.sub("(\\t)", ' ', summary).lower()
        new_text=re.sub("(\\r)", ' ', new_text)
        new_text=re.sub("(\\n)", ' ', new_text)
        new_text=re.sub("(__+)", ' ', str(new_text)).lower()   #remove _ if it occors more than one time consecutively
        new_text=re.sub("(--+)", ' ', str(new_text)).lower()   #remove - if it occors more than one time consecutively
        new_text=re.sub("(~~+)", ' ', str(new_text)).lower()   #remove ~ if it occors more than one time consecutively
        new_text=re.sub("(\+\++)", ' ', str(new_text)).lower()   #remove + if it occors more than one time consecutively
        new_text=re.sub("(\.\.+)", ' ', str(new_text)).lower()   #remove . if it occors more than one time consecutively

        new_text=re.sub(r"[<>()|&©ø\[\]\'\",;?~*!]", ' ', str(new_text)).lower() #remove <>()|&©ø"',;?~*!
        # Contraction mapping
        new_text = ' '.join([contraction_mapping[word] if word in contraction_mapping.keys() else word for word in new_text.split()])
        # Adding start and End token to signify beginning and end of summary
        new_text = 'ssss ' + new_text + ' aaaa'
        cleaned_summary.append(new_text)
    return cleaned_summary
```

```python
## Tokenizing, vectorizing text
text_tokenizer = Tokenizer()
text_tokenizer.fit_on_texts(cleaned_text)
x_train_seq = text_tokenizer.texts_to_sequences(cleaned_text)
x_test_seq = text_tokenizer.texts_to_sequences(cleaned_text_test)

## Tokenizing, vectorizing summary
summary_tokenizer = Tokenizer()
summary_tokenizer.fit_on_texts(list(cleaned_summary))
y_train_seq = summary_tokenizer.texts_to_sequences(cleaned_summary)
y_test_seq = summary_tokenizer.texts_to_sequences(cleaned_summary_test)
```

```python
# Padding text and vocab generation
x_train_pad = pad_sequences(x_train_seq, maxlen=max_text_len, padding='post')
x_test_pad = pad_sequences(x_test_seq, maxlen=max_text_len, padding='post')
text_vocab = len(text_tokenizer.word_index) + 1

# Padding summary and vocab generation
y_train_pad = pad_sequences(y_train_seq, maxlen=max_summary_len, padding='post')
y_test_pad = pad_sequences(y_test_seq, maxlen=max_summary_len, padding='post')
summary_vocab = len(summary_tokenizer.word_index) + 1
```

SEQ2SEQ Abstractive Model

- Sequence to sequence modelling is the basis of all summarization solution.
- In SEQ2SEQ models an encoder decoder model is formed using LSTMs
- We created our encoder-decoder model, trained it on the dataset as first step shown in the next slides

The Model

LAYERS:

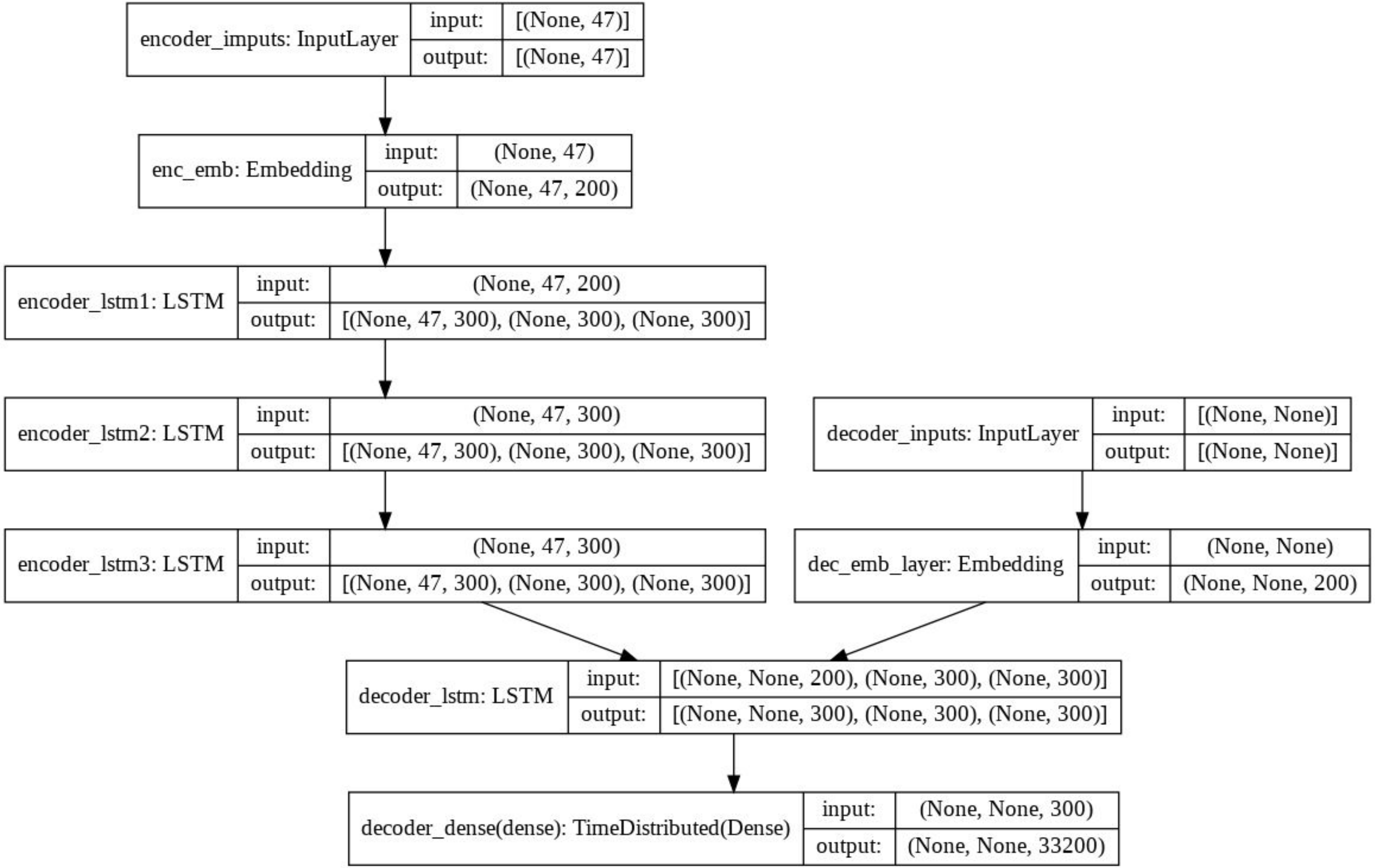
encoder:

- Encoder Embedding Layer
- 3 Encoder LSTM Layers

decoder:

- Embedding Layer
- 1 LSTM Layer
- Dense Layer

```
# Encoder
encoder_inputs = Input(shape=(max_text_len,), name = 'encoder_imputs')

#embedding layer
enc_emb =  Embedding(text_vocab, embedding_dim,trainable=True, name = 'enc_emb')(encoder_inputs)

### 2 LSTM layers for encode

#encoder lstm 1
encoder_lstm1 = LSTM(latent_dim,return_sequences=True,return_state=True,dropout=0.4,recurrent_dropout=0.4, name = 'encoder_lstm1')
encoder_output1, state_h1, state_c1 = encoder_lstm1(enc_emb)

#encoder lstm 2
encoder_lstm2 = LSTM(latent_dim,return_sequences=True,return_state=True,dropout=0.4,recurrent_dropout=0.4, name= 'encoder_lstm2')
encoder_output2, state_h2, state_c2 = encoder_lstm2(encoder_output1)

#encoder lstm 3
encoder_lstm3=LSTM(latent_dim, return_state=True, return_sequences=True,dropout=0.4,recurrent_dropout=0.4, name= 'encoder_lstm3')
encoder_output, state_h, state_c= encoder_lstm3(encoder_output2)

### Decoder

# using `encoder_states` as initial state.
decoder_inputs = Input(shape=(None,), name= 'decoder_inputs')

#embedding layer
dec_emb_layer = Embedding(summary_vocab, embedding_dim,trainable=True, name= 'dec_emb_layer')
dec_emb = dec_emb_layer(decoder_inputs)

decoder_lstm = LSTM(latent_dim, return_sequences=True, return_state=True,dropout=0.4,recurrent_dropout=0.2, name='decoder_lstm')
decoder_outputs, decoder_fwd_state, decoder_back_state = decoder_lstm(dec_emb,initial_state=[state_h, state_c])

#dense layer
decoder_dense =  TimeDistributed(Dense(summary_vocab, activation='softmax'), name= 'decoder_dense')
decoder_outputs = decoder_dense(decoder_outputs)

# Define the model
model = Model([encoder_inputs, decoder_inputs], decoder_outputs)

model.summary()
```


encoder_imputs: InputLayer
input:
[(None, 47)]
output:
[(None, 47)]
enc_emb: Embedding
input:
(None, 47)
output:
(None, 47, 200)
encoder_lstm1: LSTM
input:
(None, 47, 200)
output:
[(None, 47, 300), (None, 300), (None, 300)]
encoder_lstm2: LSTM
input:
(None, 47, 300)
output:
[(None, 47, 300), (None, 300), (None, 300)]
decoder_inputs: InputLayer
input:
[(None, None)]
output:
[(None, None)]
encoder_lstm3: LSTM
input:
(None, 47, 300)
output:
[(None, 47, 300), (None, 300), (None, 300)]
dec_emb_layer: Embedding
input:
(None, None)
output:
(None, None, 200)
decoder_lstm: LSTM
input:
[(None, None, 200), (None, 300), (None, 300)]
output:
[(None, None, 300), (None, 300), (None, 300)]
decoder_dense(dense): TimeDistributed(Dense)
input:
(None, None, 300)
output:
(None, None, 33200)

We then had to separate the encoder and decoder layers in order to summarize sequences:

SEQ2SEQ Enocoder

```
encoder_model = Model(inputs = encoder_inputs, outputs = [encoder_output,state_h,state_c])
encoder_model.summary()
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
encoder_imputs (InputLayer)  [(None, 47)]              0
_________________________________________________________________
enc_emb (Embedding)          (None, 47, 200)           18352400
_________________________________________________________________
encoder_lstm1 (LSTM)         [(None, 47, 300), (None,  601200
_________________________________________________________________
encoder_lstm2 (LSTM)         [(None, 47, 300), (None,  721200
_________________________________________________________________
encoder_lstm3 (LSTM)         [(None, 47, 300), (None,  721200
=================================================================
Total params: 20,396,000
Trainable params: 20,396,000
Non-trainable params: 0
_________________________________________________________________
```

SEQ2SEQ Decoder

```
decoder_state_input_h = Input(shape=(latent_dim,))
decoder_state_input_c = Input(shape=(latent_dim,))
decoder_hidden_state_input = Input(shape=(max_text_len,latent_dim))

embedded_decoder_inputs = dec_emb_layer(decoder_inputs)
decoder_outputs, decoder_state_h, decoder_state_c = decoder_lstm(
    embedded_decoder_inputs , initial_state=[decoder_state_input_h,decoder_state_input_c])
decoder_outputs2 = decoder_dense(decoder_outputs)

# Final decoder model
decoder_model = Model(
    [decoder_inputs] + [decoder_hidden_state_input,decoder_state_input_h, decoder_state_input_c],
    [decoder_outputs2] + [ decoder_state_h, decoder_state_c])
```

Summarization function and generation of word_index dicts from vocabs

```
summary_index_dict =summary_tokenizer.index_word
text_index_dice =text_tokenizer.index_word
summary_word_index=summary_tokenizer.word_index
```

```
def summarize(input_seq):
    '''
    This function produces summary sequences with the inference models generated from the trained model
    Input:
        It takes the input sequence to be summarized
    Output:
        It yields the summary sequence

    '''
    # Encode the input as state vectors.
    e_out, e_h, e_c = encoder_model.predict(input_seq)

    # Generate empty target sequence of length 1.
    summary_seq = np.zeros((1,1))

    # Populate the first word of target sequence with the start word.
    summary_seq[0, 0] = summary_word_index['ssss']

    stop_condition = False
    decoded_sentence = ''
    while not stop_condition:
        # Decoding using the states from the encoder and the decoder output fro the last instance
        output_tokens, h, c = decoder_model.predict([summary_seq] + [e_out, e_h, e_c])

        # Sample a token
        sampled_token_index = np.argmax(output_tokens[0, -1, :])
        #print(sampled_token_index)

        # get the word of the output token
        sampled_token = summary_index_dict[sampled_token_index]

        # Check if its the end token
        if(sampled_token!= 'aaaa'):
            decoded_sentence += ' '+sampled_token

        # Exit condition: either hit max length or find stop word.
        if (sampled_token == 'aaaa'  or len(decoded_sentence.split()) >= (max_summary_len-1)):
            stop_condition = True

        # Update the target sequence (of length 1).
        summary_seq = np.zeros((1,1))
        summary_seq[0, 0] = sampled_token_index

        # Update internal states
        e_h, e_c = h, c

    #print(len(output_tokens[0,-1,:]))
    return decoded_sentence
```

Results : 4000 data samples

Original Text:
The Kerala government on Tuesday said the state's tourism sector registered a loss of about ?1,000 crore post the demonetisation drive by the Centre. The number of foreign tourists visiting the state decreased by 10-15% while the number of domestic tourists dipped by 20-30%. The fall is in contrast to the increase in the numbers before demonetisation, the government added.

Generated headline:
I am blessed that you even know i exist ranveer to amitabh

Other Generated headlines:
Delhi metro to launch housing scheme with 2bhk 3bhk flats

Results : 80000 data samples

Original Text:
The Kerala government on Tuesday said the state's tourism sector registered a loss of about ?1,000 crore post the demonetisation drive by the Centre. The number of foreign tourists visiting the state decreased by 10-15% while the number of domestic tourists dipped by 20-30%. The fall is in contrast to the increase in the numbers before demonetisation, the government added.

Generated headline:
india s first lingerie vending machine to be brought down

Other headlines:
i am not a feminist i am a feminist i am scared karan johar

Individuals Contributions

As a team, the workload was equally distributed between the team members. Although the fact that each one was assigned to a specific task, continuous supervision and feedback were given from each team member for each task. Here is a summary of the workload distribution.

- Dataset preprocessing was done by **Salma Elbess.**
- TF-IDF extractive summarization by **Elsayed Mostafa.**
- TextRank extractive summarization by **Muhammed Alasmar.**
- Seq2Seq Abstractive summarization by **Asmaa M. Ibrahim.**
- Pretrained abstractive summarization models by **Salma Elbess.**
-

Thank you very much for this NLP course!

Summary: THANKS