Highlights:
News Article Summarization

Milestone: Project Report

Shivani Shrikant Naik

564-210-7537

naik.shiv@northeastern.edu

Signature: 

Submission Date: 08/05/2022

Problem Setting

With human lives getting increasingly busy day by day, it is difficult to stay in touch with everything that is happening around the world. People want to stay up to date on the latest news while spending minimal time reading it. In such a scenario, reading the summary or highlights of news stories is a more convenient way to keep up with the news. Reading short summaries of all articles saves time for those who do not want to read the complete article, and if someone needs more information after reading the summary, they can read the articles that interest them.

The goal of this project is to automate the text summarizing of news stories. The system will generate a brief synopsis of approximately 60-100 words for a news article that is provided.

Problem Definition

Given a fresh news article as input, the system will generate a text summary of it. This project will involve exploring extractive and abstractive text summarization techniques. Extractive technique generates summary using important sentences from the original

text, whereas abstractive technique summarizes most important information in a new way using advanced natural language processing.

Data Sources

The CNN / Dailymail News Dataset, which contains over 300,000 news stories and is available on both Papers with Code and Kaggle, will be used for the research.

Each article has the article body and a human generated summary. The dataset was collected for a question and answering system, and can also be used to create a text summarization system.
The CNN articles collection range from April 2007 to April 2015, whereas the DailyMail articles are between June 2010 and April 2015.

Dataset Link: https://paperswithcode.com/dataset/cnn-daily-mail-1

Data Description

The corpus is divided into train, valid and test sets. Training set has approximately 287K articles, valid set has 13K and test set 11K articles.

The original articles have 766 words and 29 sentences on average. The summaries for these articles have 56 words and 3.7 sentences on average.

For every article, the dataset has:
ID: String ID of article
Article: Entire article body of ~766 words
Highlights: Summary of the article of ~56 words

This data can be used to train extractive as well as abstractive text summarization models using deep learning models.

Exploratory Data Analysis

The data is divided into training, validation and test sets. Each article has a human generated summary associated with it.

- Training set: 287K articles

- Validation set: 13K articles
- Test set: 11K articles

Fields:

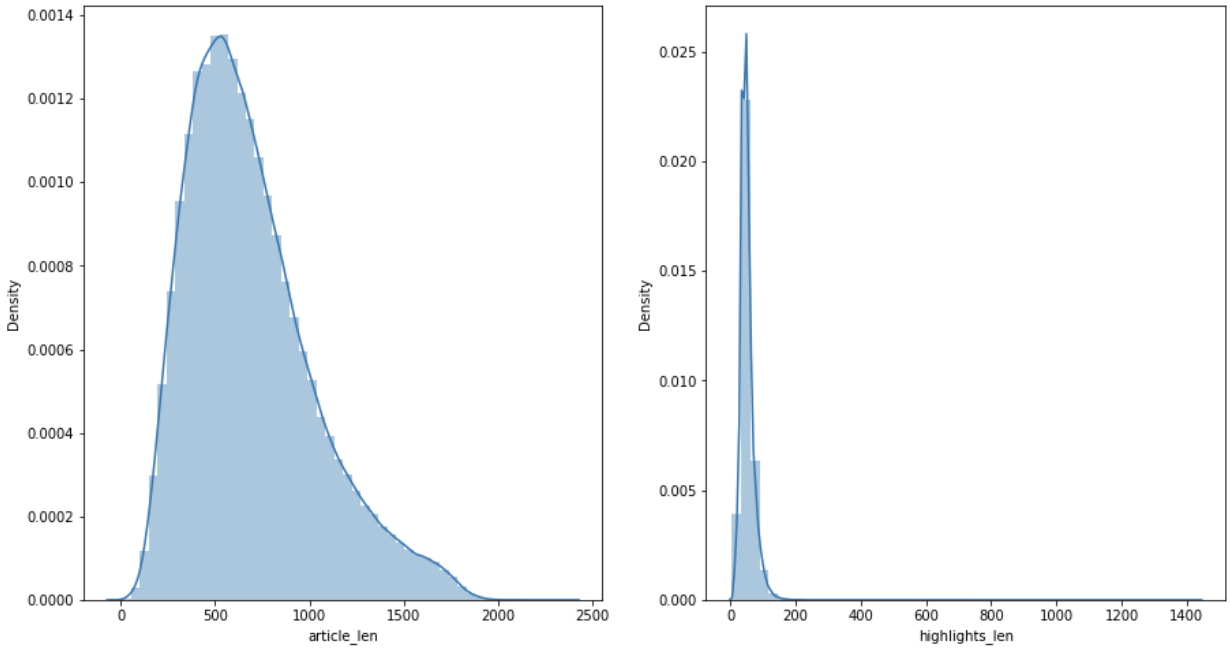
- ID : Unique ID of article
- Article: News article body
- Highlights: News article summary

Here is the snapshot of the data showcasing all fields:

	id	article	highlights
0	fef38572c5452c1a79af7cfba24ac0eaad40c0ba	By . Joshua Gardner . and Ap Reporter . A Cal...	Robert Cox of Manteca, California was just pas...
1	d222143d15cc93b65d7f76500e878a290f9b85a4	Sinister: David Russell lured his 19-year-old ...	David Russell has been jailed for life for kid...
2	cc1af28a940a9d8fe9fa2dd2f24dd9b9cc916a43	By . Victoria Woollaston . and Jonathan O'Call...	Using modern dating techniques, .\na Cambridge...
3	7a17330f65227aa1155ec9fab9972dd35d207612	(CNN)The first indication that something was w...	All three girls skipped school and took a flig...
4	80a774435cfe17b62d5103733378d0a94b4663d5	The Islamic State terror group has issued batt...	Terror group has published safety advise for i...

Distribution of Article and Summary Length

Doing analysis of the article and summary length will help in deciding model parameters. For this purpose, the distribution of article and summary length can be examined.

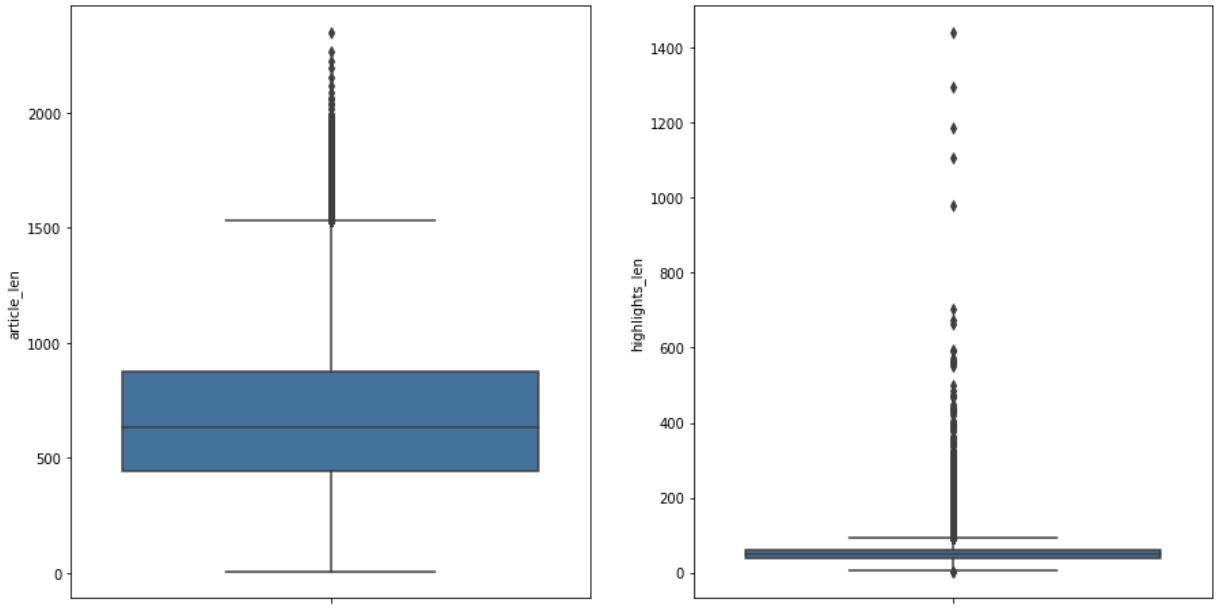


From the length distributions we can see that articles have a median length of 700 words and summaries have a median length of 60.

Both the distributions are right skewed, indicating that most of the articles have length less than or equal to 700, with a few articles having length greater than 1600 words.

Box Plot of Summary and Article Length

Summary lengths and article lengths both have a few outliers, indicating few articles have longer length and corresponding longer summaries. This can be verified with the below box plots. Both indicate that outliers are towards the higher word length side.



Word cloud of Summaries and Articles

A word cloud helps in analyzing the most important topics and keywords that are covered in a text. Generating word clouds of articles and summaries will help in seeing the keywords of the data and seeing patterns.

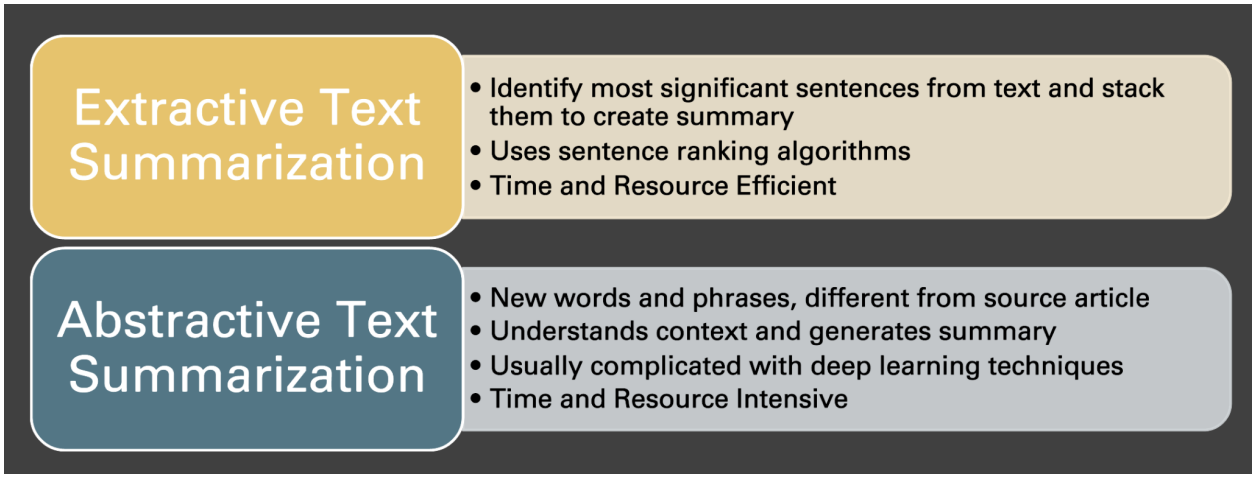
Following word clouds are plotted from a sample of news and highlights (100k). Both word clouds show similar words like year, police, official, say, new etc, so both the news articles and the summaries cover similar topics. They are also the topics that mostly appear in any news articles.

Article Word Cloud

Summary Word Cloud

Methods Used

Text summarization can be done using two ways: Extractive and Abstractive summarization.



In extractive summarization, we identify important sentences from the article and make a summary by selecting the most important sentences.

Whereas, for abstractive summarization the model understands the context and generates a summary with the important points with new phrases and language. Abstractive summarization is more similar to the way a human summarizes any content. A person might read the entire document, remember a few key points and while writing the summary, will make new sentences that include these points. Abstractive summarization follows the same concept.

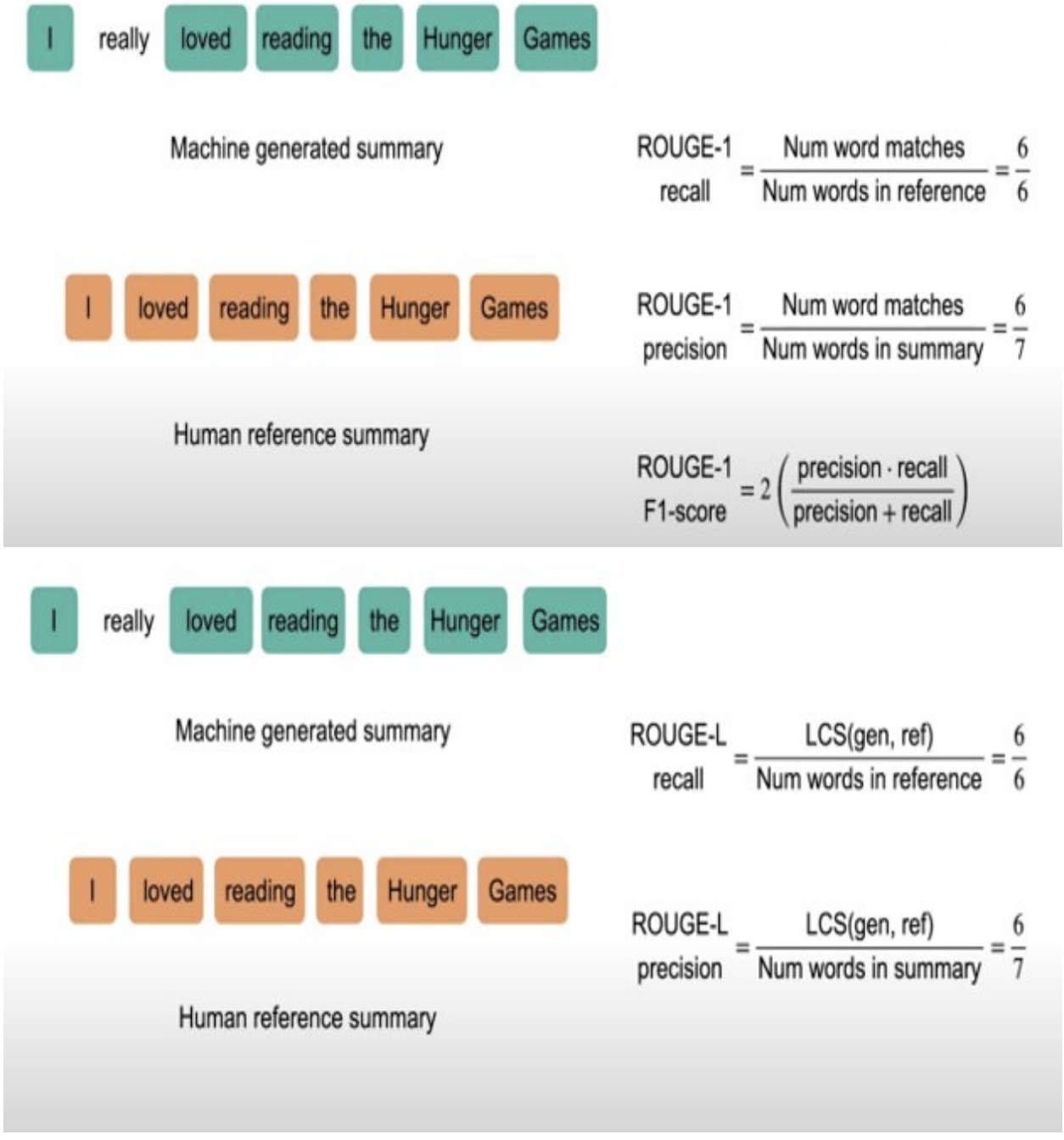
Evaluation Metric

Since the target in summarization is a text summary, it becomes complex to measure model performance. The widely used metric for text summarization is ROUGE (Recall Oriented Understudy for Gisting Evaluation). It generates a precision, recall and f1 score of the overlap between actual summary and generated summary.

ROUGE makes use of n-grams for calculating the overlap. As such, this gives rise to different ROUGE metrics like ROUGE-1, ROUGE-2, which consider 1-gram (basically word overlap) and bigrams. Another ROUGE metric is the ROUGE-L, which considers the Longest Common Subsequence (LCS) of matching words. (Code Snippets in Appendix)

Following images explain the ROUGE metric with an example of “I really loved reading th Hunger Games” as the generated summary and “I loved reading the Hunger Games” as a reference summary.

This has a ROUGE-1 recall of 100%, ROUGE-1 precision 85%, ROUGE-1 F1 Score 91% ROUGE-L recall 100% and ROUGE-L precision 85% as explained below.



This project evaluates extractive and abstractive summarization on the basis of different ROUGE scores achieved.

Following table states the SOTA T5 metrics on the CNN/Dailymail Dataset

Metric	Score
ROUGE-1	43.52
ROUGE-2	21.55
ROUGE-L	40.69

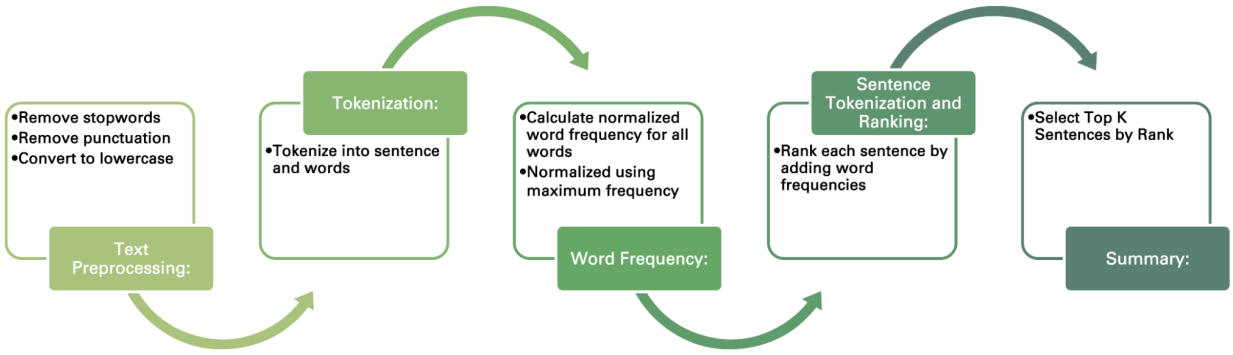
Extractive Summarization Technique

One of the techniques of extractive summarization is text ranking. In this method, each sentence of the document is assigned a score based on its importance. The top K sentences from these are selected as a summary.

Gensim provides a summarizer that performs such summarization. But, for the purpose of this project, I have implemented the extractive summarizer from scratch using python in a News Summarizer class (Code Snippets in Appendix).

Algorithm

The idea is that sentences that have words that are more frequently used throughout the article are more important. This works because we remove stopwords. For example, if an article talks about NASA, it will have many references to space, stars etc, thus these sentences will be more important.



The steps followed for extractive summarization are as follows

- Text Preprocessing (Code Snippets in Appendix)
    - Convert to lowercase

    - Remove punctuation
    - Remove stopwords
- Tokenization:
    - Tokenize into sentence and words
- Word Frequency:
    - Calculate word frequency for all words
    - Normalized using maximum frequency: Normalization helps to reduce the effect of article length
- Sentence Tokenization and Ranking:
    - Rank each sentence by adding word frequencies
- Summary:
    - Select Top K Sentences by Rank. K can be a parameter that depends on the user

Advantages of Extractive Summarization

- Time Efficient: It is very quick to perform this summarization as it does not require any complex calculations and operations
- Resource Efficient: A normal CPU works just fine for this type of summarization. We do not need heavy compute machines like GPU
- Easier to interpret summarization process
- Does not require large amount of data

Disadvantages of Extractive Summarization

- The summary generated is basically important sentences from the article. This might not cover the entire context of the article and might miss important points that are covered in discarded sentences
- Summary might look disconnected and less fluent as it is made of different sentences

ROUGE Performance on Test Set

Metric	Score
ROUGE-1	28.91
ROUGE-2	12.04

ROUGE-L	17.76
ROUGE-Lsum	24.14

Extractive summarization has a ROUGE-1 of 28.91. This means it has an overlap F1 Score of 28.91% for unigrams. This is a decent score as compared to other summarizers and SOTA models.

Abstractive Summarization Technique

Abstractive summarization generates a paraphrased summary of a given article. The model understands the context and rewrites a summary from the article and understood context. This is closer to the way a human generates a summary.

Abstractive summarization is a difficult task because the model needs to understand the semantics of a language to be able to generate new sentences that sound fluent and are grammatically correct.

Most of the models used for abstractive summarization are some kind of neural networks as deep learning models are very good at learning and generating natural language. We use an encoder decoder model to perform this task.

For this project, I have used the T5 Text-to-Text Transfer Transformer model.

Advantages of Abstractive Summarization

- Produces information rich and less redundant summary
- More human-like summary generation
- Captures important context properly
- Fluent and natural summary

Disadvantages of Abstractive Summarization

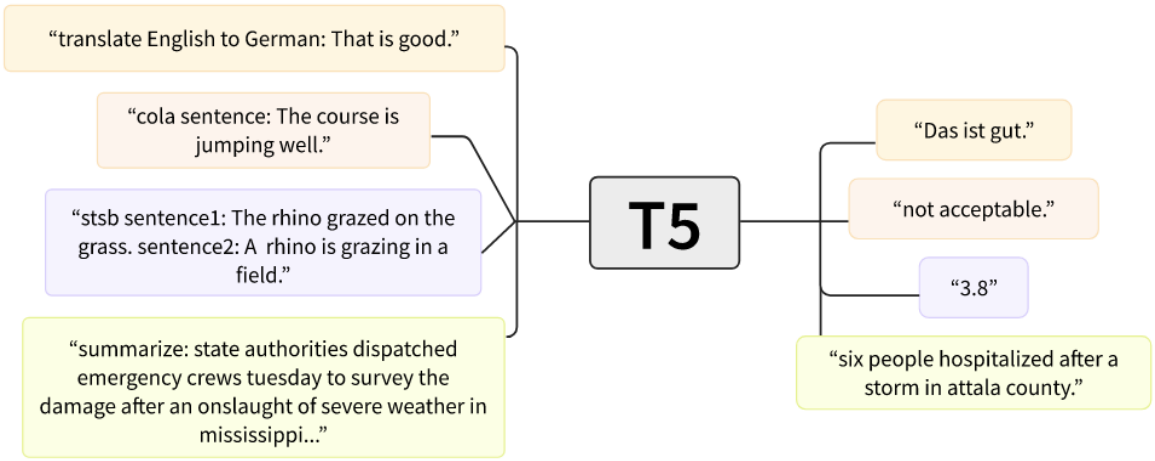
- Computationally expensive, requires accelerated hardware like GPU, TPU
- Inference also requires GPU for faster output
- Time consuming training process
- Large amount of data required for training to avoid overfitting
- Complex models like Transformers

- Harder to interpret models

T5 Transformer

The T5 Text-to-Text Transfer Transformer Model was proposed by Google in [1]. This proposes a unified framework to convert all NLP problems into a text-to-text format. This means, one model will be able to translate, summarize, classify etc based on the input prefix.

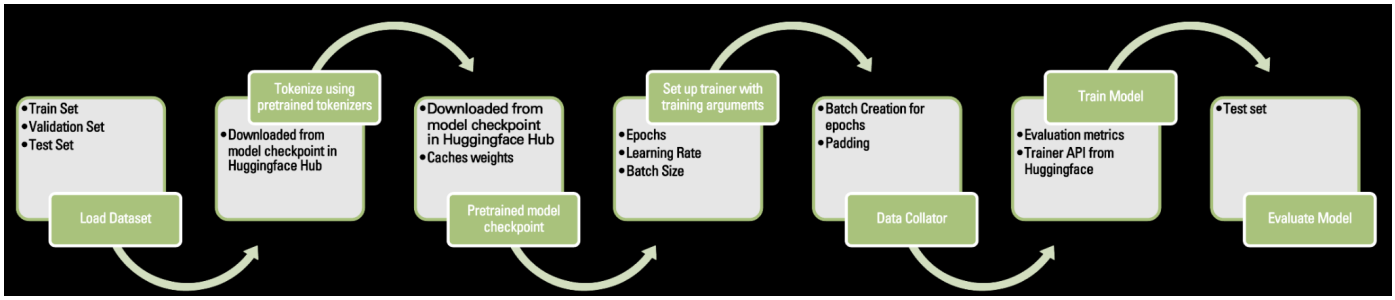
The model recognises the task by the input prefix and gives the output as a “text”. The T5 model was trained on the C4 (Colossal Clean Crawled Corpus). This dataset was obtained by web scraping and removing HTML tags. The dataset was cleaned by applying various filters like removing offensive words, removing duplicates etc and T5 was trained on this cleaned dataset.



Huggingface

Huggingface provides access to thousands of pre-trained models that can be downloaded and used for multiple tasks. These models can also be finetuned to a particular dataset and provide transfer learning.

For this project, the T5 transformer from Huggingface hub was fine tuned on the CNN/Dailymail dataset. The process to finetune a model is as follows (Code Snippets for steps in Appendix):



1. **Load dataset:** Huggingface has a repository of many datasets. The CNN/Dailymail dataset is available in this. The dataset can be downloaded and it is organized into a training set, validation set and test set.
2. **Pretrained Tokenizer:** Each pretrained model comes with its pretrained tokenizer in Hugghingface hub. This tokenizer is able to tokenize the data into the way the model expects it. The dataset is tokenized using the T5 tokenizer.
3. **Pretrained Model**: Huggingface provides pre-trained model checkpoints of all models. This has the model architecture and the trained weights that can be used for any NLP task. This model is downloaded and the weights can be fine-tuned for our specific news summarization task.
4. **Training Arguments:** Every model has various training hyperparameters that can be tuned. These arguments decide the training run of the model.
5. **Data Collator:** Huggingface provides a data collator for each NLP task. The data collator efficiently creates batches for each epoch and also pads the input.
6. **Train model:** The model is finetuned with the help of the Trainer API and the predefined training arguments
7. **Evaluate Model:** The model is evaluated on the test set using ROUGE metrics.

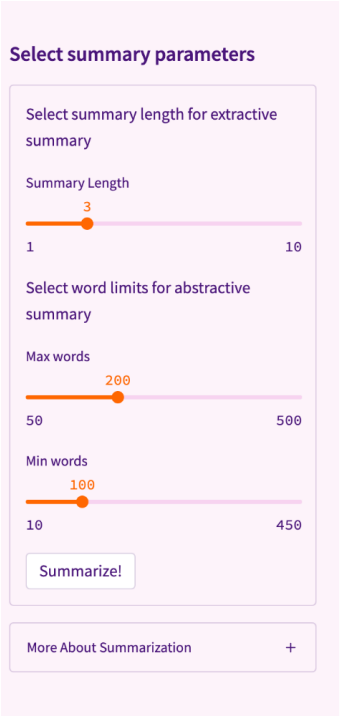
Rouge Performance on Test Set

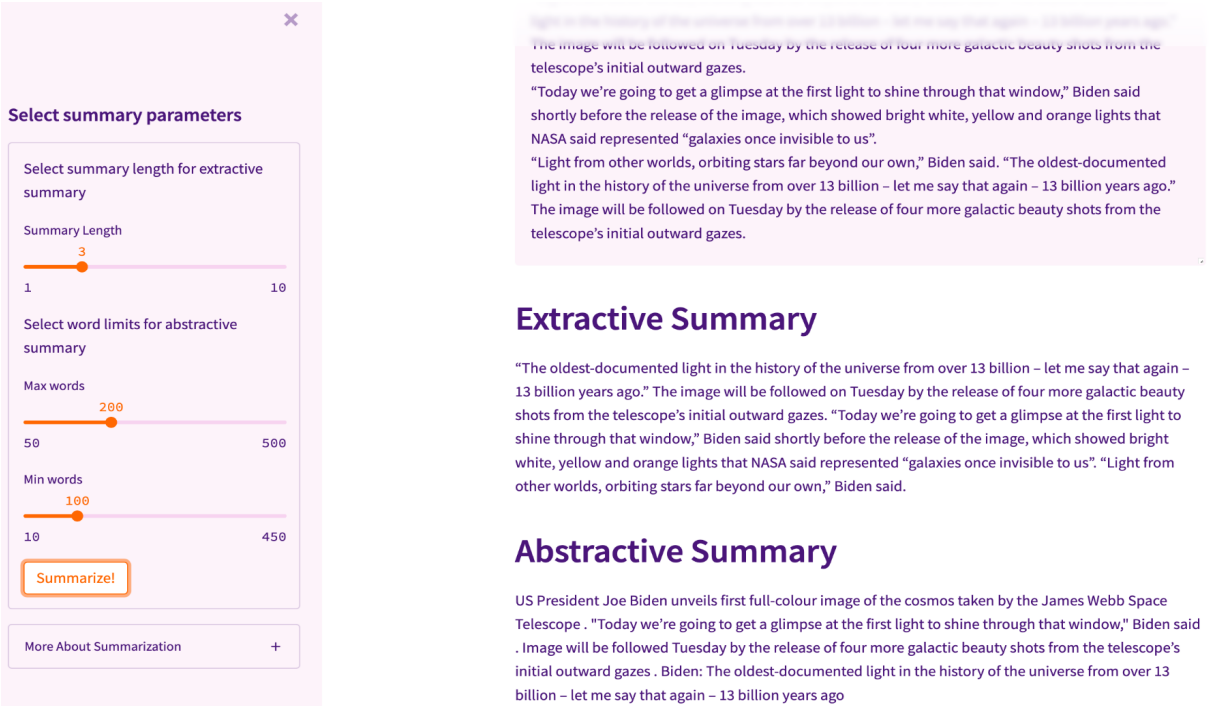
Metric	Score
ROUGE-1	38.74
ROUGE-2	17.24
ROUGE-L	26.73
ROUGE-Lsum	32.71

Deployed Application

I have created a Streamlit application that requires minimal user input in order to maintain a simple and user-friendly user interface. The user can select the summary length that they want and enter the article text in the input.

The application will generate extractive as well as abstractive summaries for the user. (Code in Appendix)

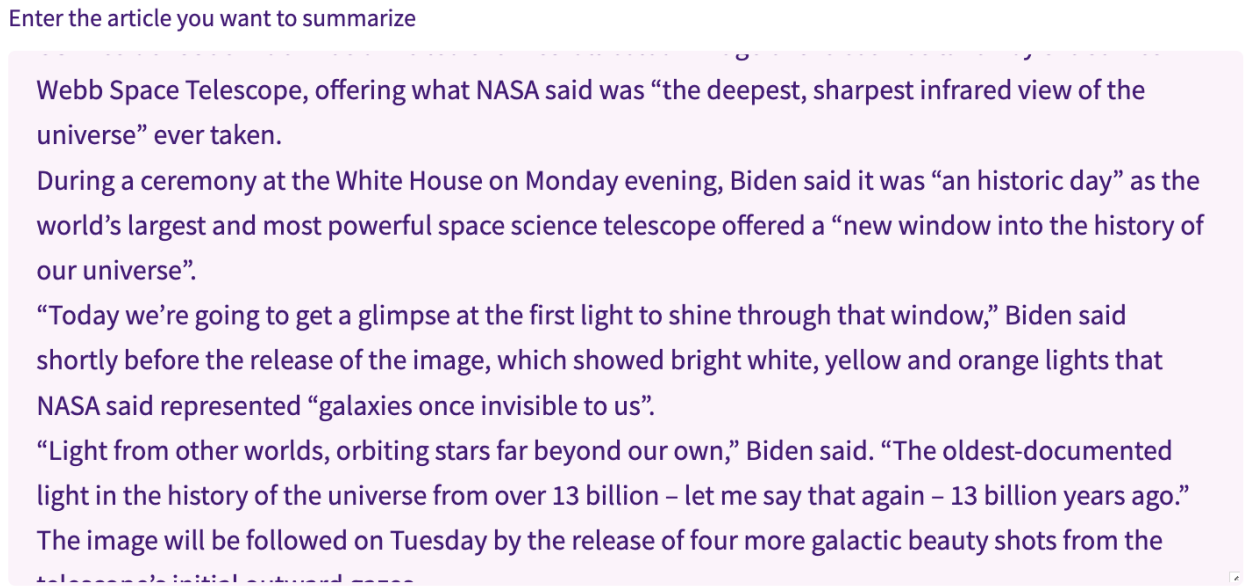




The screenshots above show the deployed service and the output for an article submitted.

Experimental Results and Model Comparison

Example article about the James Webb Telescope from NASA that was provided to the summarizer is given below, along with the generated extractive and abstractive summaries.



Extractive Summary

"The oldest-documented light in the history of the universe from over 13 billion – let me say that again – 13 billion years ago." The image will be followed on Tuesday by the release of four more galactic beauty shots from the telescope's initial outward gazes. "Today we're going to get a glimpse at the first light to shine through that window," Biden said shortly before the release of the image, which showed bright white, yellow and orange lights that NASA said represented "galaxies once invisible to us". "Light from other worlds, orbiting stars far beyond our own," Biden said.

Abstractive Summary

US President Joe Biden unveils first full-colour image of the cosmos taken by the James Webb Space Telescope . "Today we're going to get a glimpse at the first light to shine through that window," Biden said . Image will be followed Tuesday by the release of four more galactic beauty shots from the telescope's initial outward gazes . Biden: The oldest-documented light in the history of the universe from over 13 billion – let me say that again – 13 billion years ago

It can be observed that the abstractive summary is much more real-like. It covers all important points and also mentions the name "James Webb Space Telescope". Whereas extractive summary only selects important sentences from the original article.

Both summaries do capture the key points covered in the news article.

Now, to compare the ROUGE metrics

Metric	Extractive Summarizer	Abstractive Summarizer	SOTA T5 Summarizer
ROUGE-1	28.91	38.74	43.52
ROUGE-2	12.04	17.24	21.55
ROUGE-L	17.76	26.73	40.69
ROUGE-Lsum	24.14	32.71	

- Abstractive summarizer performs better in terms of all types of ROUGE scores than extractive summarizer

- Extractive summarizer is better in terms of resource and time consumption, whereas abstractive summarization requires a lot of compute power for training as well as inference

- That being said, if the model is deployed on the server with GPU availability, abstractive summarizer inference would be fairly quick

- Considering all these points and performance, abstractive summarization model with a T5 transformer is the better model

Conclusion

This project will help users in staying up to date with the latest news. With lives getting busier everyday, people are always looking for things that would help them in improving their productivity and efficiency. This application will be a small step in bridging the gap between people who want to know about world stories but do not want to invest a lot of time and the happenings around the world.

Model comparison conclusion

After trying multiple examples and looking at the results listed in the experimental results section, it can be concluded that an abstractive summarizer creates more human-like summaries than an extractive summary and is a better summarizing model.

Things learnt

- Exploratory data analysis techniques to understand more about data
- Extractive summarization techniques to summarize text
- Abstractive summarization techniques and fine tuning models
- Working of T5 transformer and Huggingface for fine tuning
- Creating a Streamlit application with a trained machine learning model that users can easily interact with

Future Improvements

1. **More Data:** To further improve the system, the first enhancement could be to train the model with more data. Due to resource constraints, the model was trained on a subset of data. Including more data will help the model in creating more accurate summaries.

2. **Newsletter**: Another improvement could be making a weekly newsletter of important highlights. Users could subscribe to this newsletter and get updates about the most important news of the week in a short format.
3. **User Preference:** Next enhancement could be sending the news that a particular user is interested in. For instance, not everyone would be interested in politics, so it makes sense to make recommendations of news summaries the user could be interested in.

Appendix

Preprocessing class for Extractive summarization:
Provides functions for lowercasing, stopword removal, punctuation removal and other preprocessing tasks.

```
class Preprocess():
  def __init__(self):
    pass
  def toLower(self, x):
    '''Converts string to lowercase'''
    return x.lower()

  def sentenceTokenize(self, x):
    '''Tokenizes document into sentences'''
    sent_tokenizer = nltk.data.load("tokenizers/punkt/english.pickle")
    sentences = sent_tokenizer.tokenize(x)
    return sentences

  def preprocess_sentences(self, all_sentences):
    '''Tokenizes sentences into words, removes punctuations, stopwords and
    performs tokenization'''
    word_tokenizer = nltk.RegexpTokenizer(r"\w+")
    sentences = []
    special_characters = re.compile("[^A-Za-z0-9 ]")
    for s in all_sentences:
      # remove punctuation
      s = re.sub(special_characters, " ", s)
      # Word tokenize
      words = word_tokenizer.tokenize(s)
      # Remove Stopwords
      words = self.removeStopwords(words)
      # Perform lemmatization
      words = self.wordnet_lemmatize(words)
      sentences.append(words)
    return sentences

  def removeStopwords(self, sentence):
    '''Removes stopwords from a sentence'''
    stop_words = stopwords.words('english')
    tokens = [token for token in sentence if token not in stop_words]
    return tokens

  def wordnet_lemmatize(self, sentence):
    '''Lemmatizes tokens in a sentence'''
    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(token, pos='v') for token in sentence]
    return tokens
```

```
def complete_preprocess(self, text):
  '''Performs complete preprocessing on document'''
  #Convert text to lowercase
  text_lower = self.toLower(text)
  #Sentence tokenize the document
  sentences = self.sentenceTokenize(text_lower)
  #Preprocess all sentences
  preprocessed_sentences = self.preprocess_sentences(sentences)
  return preprocessed_sentences

def generate_wordcloud(self, text):
  word_cloud = WordCloud(collocations = False, background_color = 'white').generate(text)
  plt.figure(figsize=(15,8))
  plt.imshow(word_cloud, interpolation='bilinear')
  plt.axis("off")
  plt.show()

def calculate_length(self, df):
  df["article_len"] = df["article"].apply(lambda x: len(x.split()))
  df["highlights_len"] = df["highlights"].apply(lambda x: len(x.split()))
  return df
```

News Summarizer class for extractive summarization:
Contains summarization code, ROUGE evaluation.

```
class NewsSummarization():
  def __init__(self):
    pass
  def extractive_summary(self, text, sentence_len = 8, num_sentences = 3):
    '''Generates extractive summary of num_sentences length using sentence scoring'''
    word_frequencies = {}
    # Instantiate Custom Preprocessor class
    preprocessor = Preprocess()
    # preprocess and tokenize article
    tokenized_article = preprocessor.complete_preprocess(text)
    #calculate word frequencies
    for sentence in tokenized_article:
      for word in sentence:
        if word not in word_frequencies.keys():
          word_frequencies[word] = 1
        else:
          word_frequencies[word] += 1
    #get maximum frequency for score normalisation
    maximum_frequency = max(word_frequencies.values())
    #normalize word frequency
    for word in word_frequencies.keys():
          word_frequencies[word] = (word_frequencies[word]/maximum_frequency)
    sentence_scores = {}

    # score sentences by adding word scores
    sentence_list = nltk.sent_tokenize(text)
    for sent in sentence_list:
        for word in nltk.word_tokenize(sent.lower()):
            if word in word_frequencies.keys():
                if len(sent.split(' ')) > sentence_len:
                    if sent not in sentence_scores.keys():
                        sentence_scores[sent] = word_frequencies[word]
                    else:
                        sentence_scores[sent] += word_frequencies[word]
    # get sentences with largest sentence scores
    summary_sentences = heapq.nlargest(num_sentences, sentence_scores, key=sentence_scores.get)
    # join and get extractive summary
    summary = ' '.join(summary_sentences)
    return summary

  def get_rouge_score(self, actual_summary, generated_summary):
    scorer = rouge_scorer.RougeScorer(['rouge1', 'rougeL'], use_stemmer=True)
    scores = scorer.score(actual_summary, generated_summary)
    return scores
```

Evaluation code for extractive and abstractive summarization:

Computes ROUGE metrics for test set

```
def compute_metrics(eval_pred):
    '''Computes metrics that can be used while training'''
    predictions, labels = eval_pred
    decoded_preds = tokenizer.batch_decode(predictions, skip_special_tokens=True)
    # Replace -100 in the labels as we can't decode them
    labels = np.where(labels != -100, labels, tokenizer.pad_token_id)
    # Decode reference summaries into text
    decoded_labels = tokenizer.batch_decode(labels, skip_special_tokens=True)
    # newline after each sentence
    decoded_preds = ["\n".join(sent_tokenize(pred.strip())) for pred in decoded_preds]
    decoded_labels = ["\n".join(sent_tokenize(label.strip())) for label in decoded_labels]
    # Compute ROUGE scores
    result = rouge_score.compute(
        predictions=decoded_preds, references=decoded_labels, use_stemmer=True
    )
    # median scores, we get the fmeasure as percentage
    result = {key: value.mid.fmeasure * 100 for key, value in result.items()}
    return {k: round(v, 4) for k, v in result.items()}
```

```
  def evaluate_extractive(self, dataset, metric):
    summaries = [self.extractive_summary(text) for text in dataset["article"]]
    score =  metric.compute(predictions=summaries, references=dataset["highlights"])
    rouge_names = ["rouge1", "rouge2", "rougeL", "rougeLsum"]
    rouge_dict = dict((rn, round(score[rn].mid.fmeasure * 100, 2)) for rn in rouge_names)
    return rouge_dict

  def evaluate_abstractive(self, dataset, metric, summarizer):
    summaries =  [summarizer(text, max_length = 120, min_length = 80, do_sample = False)[0]['summary_text'] for text in dataset["article"]]
    score =  metric.compute(predictions=summaries, references=dataset["highlights"])
    rouge_names = ["rouge1", "rouge2", "rougeL", "rougeLsum"]
    rouge_dict = dict((rn, round(score[rn].mid.fmeasure * 100, 2)) for rn in rouge_names)
    return rouge_dict
```

Preprocess function for T5 transformer:

Function for preprocessing dataset and tokenizing it for T5 transformer

```
def preprocess_function(examples):
  '''Preprocesses the data as required by huggingface transformer'''
    model_inputs = tokenizer(
        examples["article"], max_length=max_input_length, truncation=True
    )
    with tokenizer.as_target_tokenizer():
        labels = tokenizer(
            examples["highlights"], max_length=max_target_length, truncation=True
        )

    model_inputs["labels"] = labels["input_ids"]
    return model_inputs
```

Training Arguments for T5 transformer:

Setting up Training Arguments

Let us set up the training parameters.

- Used batch size 8 to fit model in memory and avoid OOM issues
- Epochs: Recommended to train for 5-8 epochs to prevent overfitting.
- Learning rate: Small learning rate, as we want to finetune and not train from scratch. Used 5.6 x e-4 after experimenting with multiple values
- Evaluation strategy: 'epoch', evaluates model after an epoch
- weight_decay: Used 0.01
- push_to_hub: Save model training files to huggingface hub, helpful as Google Colab can disconnect anytime
- predict_with_generate: True, to evaluate model after every epoch

```
[ ]
    batch_size = 8
    num_train_epochs = 5
    # Show the training loss with every epoch
    logging_steps = len(tokenized_datasets["train"]) // batch_size
    model_name = model_checkpoint.split("/")[-1]

    args = Seq2SeqTrainingArguments(
        output_dir=f"{model_name}-finetuned-cnn-news",
        evaluation_strategy="epoch",
        learning_rate=5.6e-4,
        per_device_train_batch_size=batch_size,
        per_device_eval_batch_size=batch_size,
        weight_decay=0.01,
        save_total_limit=3,
        num_train_epochs=num_train_epochs,
        predict_with_generate=True,
        logging_steps=logging_steps,
        push_to_hub=True,
    )
```

Streamlit App Code:

Code for creating web app and providing summarization as a service.

highlights_app.py

```
#Load model
hub_model_id = "shivaniNK8/t5-small-finetuned-cnn-news"
summarizer = pipeline("summarization", model=hub_model_id)


#Create header
st.write("""# HIGHLIGHTS! \n ### A News Summarizer""")
st.write("Provide a news article and get a summary within seconds! ")

# Image
image = Image.open('newspaper.jpeg')
st.image(image)


#Create and name sidebar
st.sidebar.header('Select summary parameters')
with st.sidebar.form("input_form"):
    st.write('Select summary length for extractive summary')
    max_sentences = st.slider('Summary Length', 1, 10, step=1, value=3)
    st.write('Select word limits for abstractive summary')
    max_words = st.slider('Max words', 50, 500, step=10, value=200)
    min_words = st.slider('Min words', 10, 450, step=10, value=100)

    submit_button = st.form_submit_button("Summarize!")


article = st.text_area(label = "Enter the article you want to summarize", height = 300, value = "Enter Article Body Here")



news_summarizer = NewsSummarization()


if submit_button:
    st.write("## Extractive Summary")
    ex_summary = news_summarizer.extractive_summary(article, num_sentences = max_sentences)
    st.write(ex_summary)

    summary = summarizer(article, max_length = max_words, min_length = min_words, do_sample = False)
    abs_summary = summary[0]['summary_text']
    st.write("## Abstractive Summary")
    st.write(abs_summary)

with st.sidebar.expander("More About Summarization"):
     st.markdown("""
        In extractive summarization, we identify important sentences from the article and make a summary by selecting the most important sentences. <br>

        Whereas, for abstractive summarization the model understands the context and generates a summary with the important points with new phrases and language.
        Abstractive summarization is more similar to the way a human summarizes any content. A person might read the entire document,
        remember a few key points and while writing the summary, will make new sentences that include these points. Abstractive summarization follows the same concept.

    """)
```

References

1. Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., & Liu, P. J. (2020, July 28). *Exploring the limits of transfer learning with a unified text-to-text transformer*. arXiv.org. Retrieved August 4, 2022, from https://arxiv.org/abs/1910.10683
2. Nallapati, R., Zhou, B., santos, C. N. dos, Gulcehre, C., & Xiang, B. (2016, August 26). *Abstractive text summarization using sequence-to-sequence RNNS and beyond*. arXiv.org. Retrieved August 4, 2022, from https://arxiv.org/abs/1602.06023v5
3. Lin, C.-Y. (n.d.). *Rouge: A package for automatic evaluation of summaries*. ACL Anthology. Retrieved August 4, 2022, from https://aclanthology.org/W04-1013/
4. community, T. H. F. D. (n.d.). *Cnn_dailymail · datasets at hugging face*. cnn_dailymail · Datasets at Hugging Face. Retrieved August 4, 2022, from https://huggingface.co/datasets/cnn_dailymail
5. *Streamlit docs*. Streamlit documentation. (n.d.). Retrieved August 4, 2022, from https://docs.streamlit.io/