

Article Summarization

Rishabh Ganesh and Navpreet Kaur

15 May 2022



1 Introduction

1.1 Problem Statement

The sponsor of our project is Red Ventures. Red Ventures is a media company that works on improving online experiences for the users of their brands through premium content, AI-driven digital marketing, and analytics [1]. Our industry liaisons were Dr. Ruiye Ni, Data Science Manager, and Jason Woo, Data Scientist.

The statement of our problem is to generate summaries no longer than 280 characters for a dataset comprised of 1177 articles from Red Ventures websites, and generate article recommendations based on the summaries produced. Often when we search for something using an online search engine, the engine provides search results that have links to websites containing the best matches to the search query according to the algorithm that the search engine uses. Under these links, most search engines provide brief summaries of the content that users may find by clicking on the corresponding link. Since we worked on article summarization for Red Ventures, one primary goal of our problem was to increase the quality of the summaries for the articles that appear on Red Ventures' websites so that these websites appear higher in search results when users enter a reasonably related search query.

In terms of article recommendations, it is important to note that websites where articles are published have increasingly been following the trend of having a second, similar article appear directly below the article on the same webpage where users are reading at a given time. As such, the other goal of our problem is to generate logical recommendations for what this second article should be using the summaries that our model generates. This problem of article summarization and recommendation is important because high quality summaries can move Red Ventures articles, or any articles, higher in search results, which

increases user traffic on Red Ventures websites and therefore generates more advertisement revenue because of the increased likelihood that a user clicks on an advertisement after visiting a website. The recommendations after a given article are of similar importance, since high quality article recommendations increase user retention after reading the article they initially searched for, making them more likely to see and click on advertisements that appear lower on the webpage near the recommended second article.

1.2 Procedure

To achieve our goal, we first wanted to ensure that we had a solid understanding of the material involved in trying to solve the problem. For this we conducted research on various mechanisms that would be involved in developing a summarizer. From there, we moved on to develop some insight on the dataset provided to us by Red Ventures and cleaned it up so that it could be implemented into our models. Then, we tested various summarizing techniques until we were able to produce the desired results that were logical and competent. With the generated summaries, we moved on to create recommendations. Once we had these two items built, we implemented them into a streamlit app to make the results as clear as possible for our minimum viable product (MVP). In building our MVP, we leveraged many libraries in Python. One of the most important was the pandas library that enabled us to efficiently work with the large dataset we were given using powerful tools like the DataFrame.

The paper is organized as follows. Section 2 covers the background information that was involved to understand the given problem and the methods that we would need to utilize to solve the problem. Section 3 goes over the steps involved in data preprocessing for our dataset so that our dataset became more efficient to work with. Section 4 goes over how the summarizer for our dataset was developed and the mechanics behind how it works. It also briefly goes over how we evaluated the performance of the produced summaries. Section 5 moves on to the development of the recommender that utilized the summaries to produce recommendations. Section 6 covers our MVP, the final UI that was produced as a result of this project. We conclude our work in Section 7.

2 Background Information

Throughout the paper we will mention some terms from NLP (Natural Language Processing), so we will define them in this section and their importance in understanding our problem.

2.1 Abstractive vs. Extractive

An abstractive summarizer summarizes some text by taking important parts of the text and logically putting them together to form a summary. Abstractive summarizers have logically intensive architectures because they are often trained to identify which words

and clauses of an input text add meaning to a summary. In this sense, a summary output through an abstractive model may augment the sentences that it processed from the original input text. On the other hand, an extractive summarizer determines the most important units of an input text, and creates a summary out of these units without augmenting them further. Basically, it checks spans of the input text to determine whether they should be part of a summary. An example of a unit of some text may be sentences or words. As such, a common extractive summarizer model will choose a few of the most important sentences from an input text, and output these sentences unchanged as a summary. Technically, characters could also be units, but most if not all extractive models determine the importance of sentences or words in a piece of text, rather than characters, to use them in a summary [2].



Figure 1: Illustration of abstractive vs. extractive

2.2 Natural Language Processing Methods

Natural Language Processing (NLP) is the study of how computers process natural languages, like speech or texts [3]. Our problem deals with NLP as we are trying to process article data, which is text data. For this we use transformers to process the text data. In order to understand transformers we need to understand the concept of attention.

2.2.1 Attention

Attention is among the first topics we had to research in order to understand how transformers work and why they would be useful in our project. Attention puts weight on individual words depending on their position in an input to determine the sequence [5]. This is important since this is how humans interact with text. For computers, before attention, the most common method for sequence-to-sequence models was recurrent networks. However, using recurrent networks failed to capture any semantics about their input and only relied on the sequence of the input. Transformers use attention in their architecture to take in input and produce output as it is the most useful way to capture information of a text [6].

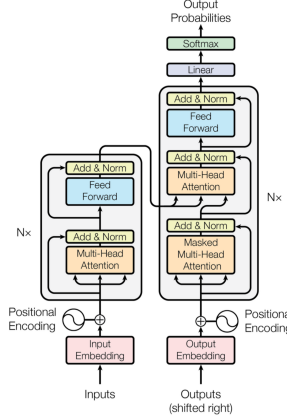


Figure 1: The Transformer - model architecture.

Figure 2: Transformer architecture. Taken from “Attention Is All You Need” [4].

2.2.2 Transformers

The transformer is the architecture working behind the scenes of our project. Transformers use the encoder and decoder together with attention mechanisms. Figure 2 shows us what the diagram for the transformer architecture looks like. We will briefly explain what is happening in this figure. The encoder takes in an input and produces embedded vectors and then uses positional encoding to determine the positions of the words in the input that are of the same size as the embedded vector. This is followed by the multi-head attention layers in both the encoder and decoder side in which the attention mechanisms are taking place. Each multi-head attention layer has the processes of the attention mechanism occurring parallel to each other. The last part of this figure is the feed-forward blocks, which is the layer where the elements of a given sequence are linearly transformed to be input for the next layer. Knowing the general high-level ideas behind how transformers work was useful in understanding open-source summarization models in this project [6, 7, 8].

2.3 Text Summarization Method

The summarization method we use is through a library called transformers, developed by Huggingface. The transformers library provides open-source access to a wide host of summarizer models that abstract the built-in architectures for research and project purposes and make the models easily iterable on various types of input that the models can handle. Using the pipeline module imported through the transformers library, we were able to use its summarizer which runs through the BART (denoising autoencoder for pretraining sequence-to-sequence models) architecture [9]. BART is a generalized version of both the BERT (bidirectional encoder representations from transformers) developed by Google and the GPT-2 (generative pre-trained transformer 2) developed by OpenAI. Basically, these architectures were published as novel ways for a computer to process natural languages by splitting text into tokens (like words, interjections, or ’s), encoding the tokens into numerical

values based on self-attention, determining numerical weights through pre-training and fine-tuning to determine how important the tokens are in the meaning of the text, and then generating a summary of variable length [10]. BART is pre-trained on CNN articles, where many of the articles if not all of the articles used for pre-training and fine tuning are written in English [9]. While many transformers architectures can only handle up to 512 tokens of input, BART can impressively encode up to 1024 tokens of input [10]. We thought BART was the best model we could use in this project because it was highly compatible with most of the text and articles that appear in our dataset, and we thought an abstractive summary optimized for iterations fit the requirements of the problem statement very well.

3 Data Preprocessing

Data preprocessing is one of the first steps we dealt with when getting hands-on with the data provided. Data preprocessing is the process in which raw data is transformed into a more readable format. The original dataset was of raw HTML data in a csv file, which made it hard to understand. Each article, along with its source and number, were assigned a single row and the values were inconsistently spread throughout the columns. This made it hard for us to understand information such as where the article started and ended and how long each article was. To deal with this we used the pandas and numpy libraries on Python.

To begin we had to understand which information can be extracted from the data. We could clearly see the article numbers and the source for each article, but other information such as the title and author's name were not easily seen or were not available. Thus, using python's `split()` function we extracted only the article number and source for each article and produced a new csv file which now held a column for the article number and a column for the source. The next step was to create a column for the article's text. Since the contents of each article were widely spread throughout each column we created a list of lists that would loop through each row and append the values of each column into a list for each row. Each list in the list of lists was the text of the article. Now that we had the article text we included this into the new csv file.

After having the data put into a readable format, we began to explore the dataset and determine what type of articles we were working with. There were some articles that were in different languages that would become a problem since our summarizer would not be able to understand them, thus we decided to remove them from the dataset. To recognize which articles were not in English, we used the `isascii()` function in Python which checks if the string has non-English characters. However, there were still some articles that had characters that were recognizable as English, but had no meaning to them. We will call these random letters special characters. To remove these values we counted how many of these special characters existed in each article text, by counting characters that were not A through Z characters, digits, or common punctuation. If the text of the article had special characters that accounted for more than 40% of the characters in the whole text, then we decided to get rid of that text since there wasn't anything we could do with these articles.

We also removed any articles that were duplicates. Now, we finally had a clean dataset that we could input into the summarizer.

Our data preprocessing ended here, however data preprocessing can involve more steps such as stemming and tokenizing. We did not have to worry about these steps since the summarizer we use contains built-in functions that would do this. The term tokenizing simply means splitting text into words or subwords [11].

4 Summarizer

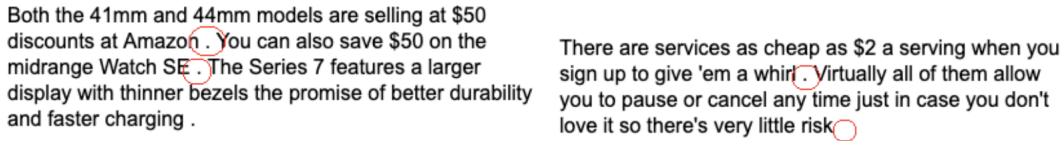
In this section, we cover the open-source text summarization model we used to generate the summaries for this project, and how we made adjustments to meet the requirements for the summaries that were specified in the problem statement.

4.1 Raw Summaries

Since the BART summarizer we imported into our Google Colab notebooks and .py files by using pipeline from the transformers library processes text through its full architecture before generating summaries, we were somewhat happy with the raw output. To leverage this model in our project, we first needed to initialize a pipeline with the parameter “summarization”. This enabled us to use the pipeline trained for summarization, so we passed in the following three parameters: an article for the text parameter, the minimum length of the summary as 15 words, and the maximum length as 55 words. We chose the maximum and minimum lengths by running the summarizer pipeline on random text that was not part of our data set and checking what maximum and minimum length would generate summaries that are close to 280 characters in length. The summarize() method in the pipeline returns a list of dictionaries, so in order to access the summary output, we use the key “summary_text” of the dictionary in the first element of the list that is returned.

Based on the output that the summarizer pipeline generated on the articles in our data set, we noticed certain things about the quality of the raw summaries. One thing we noticed about the raw output from the summarizer pipeline was that the lengths of the summaries varied. This makes sense given that the parameters we entered represented a range of possible summary lengths in words, namely between 15 words and 55 words, which is a reasonable range for summaries near 280 characters. Another reason the variable lengths of the raw summaries make sense is that the articles in our data set after preprocessing were all different, so naturally, the summaries generated by the pipeline summarizer would vary in length depending on the length of words and tokens that have pre-determined weights of importance from BART training. Some of the lengths automatically met the 280 character maximum, while others problematically exceeded the maximum. Besides this, we noticed that the summaries often ended with incomplete sentences, as some sentences at the end of summaries were missing predicates, other summaries ended with sentences looking something like the word “The” by itself, or summaries sometimes ended with dangling subordinating clauses. Additionally, there were unnecessary spaces at the end of complete

sentences between the last character of the last word in those sentences and the period at the end of the sentence. Another perhaps more serious aspect of the raw summaries that we noticed was that some of the summaries that ended on complete sentences did not have periods at the end. Given that some of these issues were apparent in the documentation and references describing how to use the BART summarizer pipeline, we knew that for the purposes of our project, we would need to add features to improve the quality of the raw summaries generated by the summarizer pipeline [12].



Both the 41mm and 44mm models are selling at \$50 discounts at Amazon . You can also save \$50 on the midrange Watch SE . The Series 7 features a larger display with thinner bezels the promise of better durability and faster charging .

There are services as cheap as \$2 a serving when you sign up to give 'em a whirl . Virtually all of them allow you to pause or cancel any time just in case you don't love it so there's very little risk .

Figure 3: Examples of raw summaries

4.2 Feature Improvements and Meeting Problem Requirements

To remedy these issues in the raw summaries, we wrote two Python functions whose purposes were to ensure that the summaries would fit the 280 character maximum and also maintain grammatical correctness. It is important to note that since the BART summarizer pipeline can take up to 1024 tokens of input (1 token is approximately 1 word), and some of the articles in our dataset were long enough to exceed 1024 tokens, we had to optimize our input for the summarizer model. After testing the model by passing in a few articles truncated to the first 1024 words into the summarizer pipeline, we noticed that the run time is long due to the high volume of the tokenized input. Because using the model to its full capacity of 1024 tokens could take up to a minute to generate one single summary, and since we needed to iterate through the entire data set of articles to generate all the summaries, we decided to make the input reasonably short to cut down run time. To do so, we made the assumption that the main idea of an article is captured in the first 350 tokens, and so we truncated each article to the first 350 words to shorten the run time of our model. The function we wrote for this is called `fit_tokens()`.

```
def fit_tokens(tx):  
    words = tx.split(" ")  
    words = words[:350]  
    tx = " ".join(words)  
    return tx
```

Figure 4: Code for `fit_tokens()`

After optimizing the input for the pipeline summarizer, we wrote a function called `fit_characters()` to ensure grammatical correctness of the summaries and to make sure our summaries were brief enough to fit the 280 character requirement. For summaries over

280 characters, we removed sentences from the end until the summary was under 280 characters. To accomplish this, we used the `split()` and `join()` methods built into Python. After removing the incomplete sentence at the end, we were left with summaries comprised of complete sentences that also fit the 280 character requirement. For missing periods at the end, we checked whether the last character of the summary was a space, and if so, we would replace it with a period using a conditional statement. To correct for spaces before periods and other small punctuation issues, we imported the `language_tool_python` library and used a tool object for English grammar checking and configured the tool object so that the spell checking feature would be turned off. The spell checker gets turned off by disabling the rule called `MORFOLOGIK_RULE` in the optional configuration parameter of the language tool object, and we do this because the autocorrection feature of the language tool diminishes the quality of the summaries because it corrects proper nouns that are not in the dictionary to the words that are closest to them in the dictionary. For example, with the spell checker on, a summary of an article about “DreamCloud” autocorrected “DreamCloud” to “Dreamland”. Since the BART abstractive summarizer pipeline does not change the tokens when they appear in the summary, there would be no spelling issues anyway, so we configured the language tool to have the spell checker turned off. Basically, we assume that the articles have no spelling errors before we run the summarizer, and since the summarizer pipeline does not change spellings, we correct all output of the summarizer using the language tool configured to disable the spell checker when a summary is returned from the `fit_characters()` function.

```
fit_characters(summ):
    tool = language_tool_python.LanguageTool('en-US', config={ 'disabledRuleIds'
    if (summ[len(summ)-1] == '.' and len(summ) <= 280):
        return tool.correct(summ)
    summ = summ[:280]
    sentences = summ.split(".")
    sentences.pop()
    newsum = '.'.join(sentences)
    if newsum[-1] == ' ':
        newsum = newsum[:len(newsum)-1]+ "."
    return tool.correct(newsum)
```

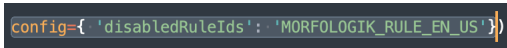


Figure 5: Code for `fit_characters()`

4.3 Quantifying Metrics

Once the summarizer returned results that were acceptable, we decided to use a similarity metric to quantify the generated summaries. By doing so we would be able to determine whether or not the summaries were good instead of having to skim through each one of them in a large dataset. It is better to have a numerical value to indicate the performance.

For this, we used the ROUGE library in Python. ROUGE stands for Recall-Oriented Understudy for Gisting Evaluation. We chose this metric as it was understandable and easy to implement into our model. It may not have been the best metric, but it provided us with sufficient information about our summaries.

ROUGE is a set of metrics [13]; for our model it consisted of three different metrics, R-1, R-2, and R-L. The 'R' stands for ROUGE, the number or letter that follows the dash tells us the length of the tokens we are comparing. R-1 measures the number of matching unigrams (single tokens), R-2 measures the number of matching bigrams, and R-L measures the longest common subsequence [13]. Suppose we consider the following string: "The sun sets in the west". If we broke this up in unigrams we would have ["The", "sun", "sets", "in", "the", "west"] and in bigrams we would have ["the sun", "sun sets", "sets in", "in the", "the west"]. For the longest common subsequence, the comparison would be made between two strings and find the longest sequence of tokens that is similar in both strings. ROUGE uses 3 measures: recall, precision and F1-score. Below we can see how each of these measures are calculated [13].

$$\text{Recall} = \frac{\# \text{ of n-grams in model and reference}}{\# \text{ of n-grams in reference}},$$

$$\text{Precision} = \frac{\# \text{ of n-grams in model and reference}}{\# \text{ of n-grams in model}},$$

$$\text{F1-Score} = 2 * \frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}}$$

Since the size of the articles was much larger than the size of the summaries, we knew that the recall value would be small. The precision on the other hand was very close to 1 for all our values, indicating that almost all of the tokens in the summary matched the tokens from the text. This told us that almost all of the words were coming from the text itself. The F1-score combined the results from the recall and precision, so in certain cases it was not very high. But again since we were able to see that most if not all of the tokens from the summaries were derived from the articles themselves, we were satisfied with the produced summaries.

5 Recommender

The next part of the problem was to create a recommender that would take in summaries as inputs and recommend which article should be read next by the user. We often see this when reading an article online and a recommendation is provided for what should be read next. To do this, we decided to implement soft-cosine similarity.

To understand soft-cosine similarity, we will briefly review cosine similarity. Cosine similarity measures the cosine of the angle between two vectors. When we input two strings that we want to compare, we first need to convert these values to numerical forms so that they can be compared by the computer. We used an open-source soft-cosine code

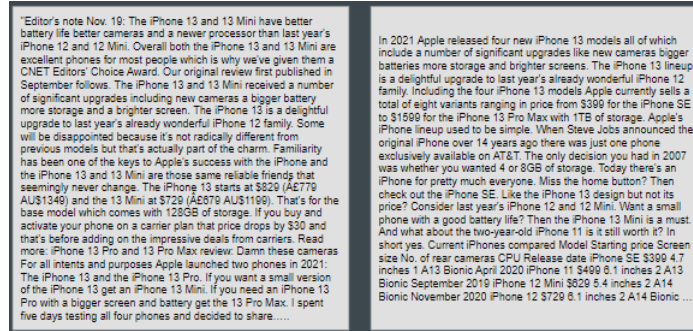


Figure 6: Article on the left is the article the user picked, article on the right shows us the recommended article.

that utilized the gensim library. The gensim library did a lot of these steps using pre-built functions. We used the `corpora.Dictionary()` function to preprocess the strings of the summaries and produce a corpora of dictionary values holding the word counts of each word. The next step was to create a similarity matrix. The difference between cosine and soft-cosine similarity is that cosine similarity is not able to check for any semantics; it does not recognize words that have the same meaning. Soft-cosine similarity, on the other hand, does this, therefore we decided to use this method. The code uses a pretrained model “fasttext-wiki-news-subwords-300” that trains which words are similar, so we do not need to do so in the code. Then we can generate a similarity matrix that compares every summary with each other from our dataset. With the created similarity matrix, we can simply check which article produces the highest score for each summary. The more similar the summaries, the higher the score [14].

6 Final UI

As a deliverable for this project, we decided to make a User Interface (UI) using the streamlit library in Python. The streamlit library is written to enable Python users to quickly make a front-end for their data science projects, so we thought it was perfectly applicable here. To access our UI, visit <https://github.com/nkaur15/ArticleSummarizationFordham> and follow the instructions. Our UI is a streamlit app which essentially enables users to locally run our project on a webpage environment. The streamlit app is comprised of five .py files that run together using the “streamlit run” command on a Linux command line. The structure of our UI is split into three tabs: a project tab, a demo tab, and a full dataframe tab. The default tab that appears as soon as the streamlit app runs is the project tab, which contains a brief description of our data set and the problem statement, information about the summarizer model we used, and the `fit_characters()` function verbatim for users to get a view of one of the major features we added to the summarizer for this project. The demo tab is where users get to pick one of the 731 articles in the trimmed data set, watch it get summarized in real time, and

Article Summarization Project

Red Ventures Article Dataset

The project we worked on is about summarizing a dataset of articles and generating article recommendations from the same dataset. This dataset was provided in the problem statement. Originally, the dataset consisted of 1176 articles, many of which were mainly comprised of special characters, so through data preprocessing we have trimmed the dataset considerably into 731 articles. Below is a dataframe with the articles numbered, along with other information like sources and text length.

	Article_num	Source	Text	len_text	num_spec_chars
0	0	CNET	Deal Savings Price Show ...	10,323.0000	98.0000
1	1	CNET	"At some point or anothe...	17,115.0000	241.0000
2	2	CNET	"The holiday shopping se...	8,578.0000	182.0000
3	3	CNET	"Looking to upgrade your...	27,064.0000	444.0000
4	4	CNET	"Editor's note Nov. 19: Th...	11,399.0000	68.0000
5	5	CNET	Josh Goldman/CNET The...	5,913.0000	77.0000
6	6	CNET	Editor's note Nov. 18: The...	13,877.0000	97.0000
7	7	CNET	Chris Monroe/CNET Goog...	7,348.0000	88.0000
8	8	CNET	"Scott Stein/CNET I'd love...	14,219.0000	147.0000
9	9	CNET	"When Samsun announce...	10,516.0000	69.0000

Summarization method

We used the summarizer from the pipeline imported through the transformers library to generate summaries for each article. After looking at open-source projects on GitHub, we thought this was a great abstractive text summarization model. The below line with the specified parameters generates summaries in the ballpark of 280 characters.

```
summary = summarizer(text, max_length=55, min_length = 15)[0]['summary_text']
```

fit_characters() is a function we wrote to ensure that the summaries fit the 280 character maximum, while also ensuring that they end on a complete sentence.

Summarization method

Article Summarization Demo

Let's run the model

Which article would you like to summarize?



Chosen article

GE Lighting announced a smart thermostat and 11 lighting products at CES 2022 on Tuesday. The Cync Smart Thermostat costs \$120 and will be available this month at Amazon Best Buy and Lowe's.

Summary

GE Lighting announced a smart thermostat and 11 lighting products at CES 2022 on Tuesday. The Cync Smart Thermostat costs \$120 and will be available this month at Amazon Best Buy and Lowe's.

Recommended Article Similarity

"Moen CES At CES 2022 Moen is upgrading its touchless kitchen faucet with a new iteration the Smart Faucet with Motion Control. Not only is this new faucet supposed to turn the water on and off hands-free this version claims hands-free temperature control as well. Use the

Dataframe with all summaries and recommendations

	ARTICLE	SUMMARY	RECOMMENDATION
0	Deal Savings Price Show m...	There are services as cheap ...	"Iryna Veklich/Getty Images...
1	"At some point or another o...	Live TV streaming services s...	"David Katzmaier/CNET Dec...
2	"The holiday shopping seas...	The 2022 TV models will be ...	"Richard Peterson/CNET CE...
3	"Looking to upgrade your e...	The best wireless earbuds h...	The standard iPhone 13 is g...
4	"Editor's note Nov. 19: The i...	The iPhone 13 is a delightfu...	In 2021 Apple released four ...
5	Josh Goldman/CNET There ...	The Dell Inspiron 16 Plus ha...	Tesla pulls the wraps off its ...
6	Editor's note Nov. 18: The c...	The Google Pixel 6 exemplif...	Pixel 6 Pro is the best Andro...
7	Chris Monroe/CNET Google'...	Google's \$180 battery-pow...	Niantic 2022 has begun and...
8	"Scott Stein/CNET I'd love t...	The Galaxy Watch 4 has mu...	Garmin Vivomove Sport rev...
9	"When Samsun announce...	The Galaxy Z Flip 3 starts at ...	"Lori Grunin/CNET There's ...

Figure 7: Pictures of UI tabs

also see the recommendation and similarity metrics. After choosing one article with the slider, users can change the article as many times as they would like. The third tab is the full dataframe tab, displaying a table with the article input next to the summary and recommendation output. The full dataframe tab is meant to show that we have already summarized all the articles in our trimmed data set (with the exception of the articles numbered 103, 117, 547, and 632; there is an empty string issue with these articles after they are processed by the summarizer pipeline). The various tabs of the streamlit app are written using basic streamlit containers, subheaders, and titles, along with the multiapp.py file available on GitHub to format the app by putting the tabs together [15].

7 Conclusion

In this section, we will discuss overall results and concluding thoughts.

7.1 Discussion

Although the original data set was 1177 articles, many of the articles ended up being meaningless non-ASCII characters. Some of them were also in languages other than English, which our model was not trained with. We removed such articles from our data set through pre-processing, trimming it down substantially to 731 articles. Of these articles, we were able to effectively deploy the summarizer and recommender, because all but 4 of them can be accessed through the slider in the demo tab of the UI and can output valid summaries and recommendations as results.

7.2 Future Work

To improve this project further, we would start by figuring out what caused the error on articles 103, 117, 547, and 632 of the trimmed data set. When the raw summaries of these articles are passed into `fit_characters()`, the Python interpreter throws an index error when the function tries to access the last index in the summary, which could only mean that something is causing the summaries to become empty strings. After figuring out this issue, we would like to make our model compatible with the articles in our data set that were written in other languages like German and French. Although summarizing text made up of non-ASCII characters is meaningless, working with a model trained to summarize languages other than English would have enabled us to fulfill the requirements of our problem on a larger fraction of our data set. Besides these potential improvements, we would like to tweak the formatting of our UI to improve the overall aesthetic of the deliverable, like the spacing of column headers on the demo tab. We would also like to work further on the recommender so that it only recommends articles that are from the same source as the input, so that we could see similar results to the ones we see on sites like CNET, managed by Red Ventures.

8 Acknowledgements

PIC Math is a program of the Mathematical Association of America (MAA) and the Society for Industrial and Applied Mathematics (SIAM). Support is provided by the National Science Foundation (NSF grant DMS-1722275).

Industry Partner Organization: Red Ventures

Faculty Advisor: Dr. Patrick McFaddin

Industry Liaisons: Dr. Joe Tenini, Dr. Ruiye Ni, Jason Woo

References

- [1] “Red ventures,” <https://www.redventures.com/>.
- [2] P. Dubey, “Understand text summarization and create your own in python,” December 2018, <https://towardsdatascience.com/understand-text-summarization-and-create-your-own-summarizer-in-python-b26a9f09fc70>.
- [3] J. Brownless, “What is natural language processing?” 2019, <https://machinelearningmastery.com/natural-language-processing/>.
- [4] N. P. J. U. L. J. A. N. G. L. K. I. P. Ashish Vaswani, Noam Shazeer, “Attention is all you need,” 2017, <https://arxiv.org/abs/1706.03762>.
- [5] N. Agrawal, “Understanding attention mechanism: Natural language processing,” 2020, <https://medium.com/analytics-vidhya/https-medium-com-understanding-attention-mechanism-natural-language-processing-9744ab6aed6a>.
- [6] Maxime, “What is a transformer?” 2019, <https://medium.com/inside-machine-learning/what-is-a-transformer-d07dd1fbec04>.
- [7] R. Kulshrestha, “Transformers,” 2020, <https://towardsdatascience.com/transformers-89034557de14>.
- [8] S. Cristina, “The transformer model,” 2021, <https://machinelearningmastery.com/the-transformer-model/>.
- [9] R. Cheng, “Abstractive summarization using pytorch,” January 2021, <https://towardsdatascience.com/abstractive-summarization-using-pytorch-f5063e67510>.
- [10] T. Rajapakse, “Bart for paraphrasing with simple transformers,” August 2020, <https://towardsdatascience.com/bart-for-paraphrasing-with-simple-transformers-7c9ea3dfdd8c#:~:text=%2D%20BART%3A%20Denoising%20Sequence%2Dto,see%20what%20it%20all%20means>.
- [11] “Summary of the tokenizers,” https://huggingface.co/docs/transformers/tokenizer_summary.
- [12] https://huggingface.co/docs/transformers/v4.19.0/en/main_classes/pipelines#transformers/.
- [13] J. Briggs, “The ultimate performance metric in nlp,” 2021, <https://towardsdatascience.com/the-ultimate-performance-metric-in-nlp-111df6c64460>.
- [14] S. Prabhakaran, “Cosine similarity - understanding the math and how it works (with python codes),” 2018, <https://www.machinelearningplus.com/nlp/cosine-similarity/>.
- [15] P. Nihar, “Streamlit multiapps,” August 2020, <https://github.com/upraneelnihar/streamlit-multiapps/blob/master/multiapp.py>.