

HAAG NLP Summarization Week 1

Michael Bock

August 2024

1 Abstract

Recent work pre-training Transformers with self-supervised objectives on large text corpora has shown great success when fine-tuned on downstream NLP tasks including text summarization. However, pre-training objectives tailored for abstractive text summarization have not been explored. Furthermore there is a lack of systematic evaluation across diverse domains. In this work, we propose pre-training large Transformer-based encoder-decoder models on massive text corpora with a new self-supervised objective. In PEGASUS, important sentences are removed/masked from an input document and are generated together as one output sequence from the remaining sentences, similar to an extractive summary. We evaluated our best PEGASUS model on 12 downstream summarization tasks spanning news, science, stories, instructions, emails, patents, and legislative bills. Experiments demonstrate it achieves state-of-the-art performance on all 12 downstream datasets measured by ROUGE scores. Our model also shows surprising performance on low-resource summarization, surpassing previous state-of-the-art results on 6 datasets with only 1000 examples. Finally we validated our results using human evaluation and show that our model summaries achieve human performance on multiple datasets.

Link: <https://arxiv.org/pdf/1912.08777>

1.1 Brief Analysis

Abstractive summarizers are summarizers which creates summaries by generating novel text that summarizes one or more documents. On the other hand, extractive summarizers directly pull sentences from a document and concatenate them to form a summary. Pegasus is an abstractive model that creates natural language summaries of documents by generating novel text. But upon further reading, this model uses Masked Language Modeling as a pretraining task with the theory that a pretraining task closer to summarization would lead to better results. Pegasus removes full sentences from the text and then learns to regenerate them.

Originally I thought that if you masked out unimportant sentences it would lead to better results by mitigating hallucinations. However, in the paper they note that masking out more important sentences led to better results. The

logic I had only holds if you are using the pretraining task as the final task. If we instead think about the pretraining task as being close to summarization, I realize that the masking combined with the regeneration acts like an extractive-abstractive summarizer where you first mask out the important sentences you want in the summary and then use the model to string them together into a coherent summary. In a sense, pegasus seems a lot like an extractive-abstractive classifier because it uses masked language modeling rather than just generating text like a model like gpt-4 does. This makes me curious what a summary would look like if we never did downstream abstractive summary generation and instead used masked language modeling. If we were careful about which sentences are extracted and which are masked out, could a more accurate summarizer be created? Furthermore, masking removes a lot of information. What if by chance we remove something we care about? Or what if something isn't totally important but is more important than another part of the text? Could we use a denoising model instead of a masking model? Pegasus points to BART, another abstractive summarizer, as a project which involves this. Another concern I have is that this pretraining task seems really close to Retrieval Augmented Generation(RAG), but RAG doesn't require us to train a model from scratch. Is Pegasus still relevant when we can just use RAG?

In summary, Pegasus seeks to improve upon other abstractive summarizers by introducing a pretraining task that is closer to summarization than other language models used. It makes me wonder what would happen if we adapted this pretraining task to be the main task. Would this achieve better benchmarks or would human experts prefer the extractive-abstractive pretraining task to the purely abstractive pegasus model? Pegasus is also not an LLM, could increasing parameter counts(this was explored in a project called T5) improve results?

2 Scripts and Code Blocks

We have not been assigned subteams yet, so I worked on finding data and on creating a client for the clearinghouse.net API.

2.1 MultiLexSum Dataset

`hugging_face.py`:

This script downloads a dataset created by clearinghouse.net that contains pairs of legal documents and summaries of those legal documents. I also included some regex code that will split the document into lines and only keep lines of at least 80 characters to filter out titles of documents, footnotes, headers, authors, signature lines, etc.

2.2 Documentation

1. Download dataset using code provided at <https://multilexsum.github.io/>
2. select example from dataset using line 8.

```

from datasets import load_dataset
import re
# please install HuggingFace datasets by pip install datasets

multi_lexsum = load_dataset("allenai/multi_lexsum", name="v20220616")
# Download multi_lexsum locally and load it as a Dataset object

example = multi_lexsum["validation"][67] # The first instance of the dev set
print('Source Document Text:')
print(len(example["sources"])) # A list of source document text for the case
for document in example["sources"]:
    print('-' * 50)
    document = re.sub(r'\[.*.*\]', '', document)
    #document = re.sub(r'Page.*\n', '', document)
    paragraphs = re.split(r'\n', document)
    #print(paragraphs)
    #print(len(paragraphs))
    print(' '.join([paragraph for paragraph in paragraphs if len(paragraph) > 80]))
for sum_len in ["long", "short", "tiny"]:
    print(example["summary/" + sum_len]) # Summaries of three lengths

~
"hugging_face.py" 20L, 874B                                8,30    All

```

Figure 1: LexSum downloader and filterer

3. Run `python3 hugging_face.py tee case.txt`— to see and save results

2.3 ch_openapi.json

Created an openapi.json for clearinghouse.net api

2.4 Results Visualization

COMPLAINT and JURY TRIAL DEMAND

Defendant: NATURE OF THE ACTION This is a public enforcement action to correct the unlawful employment practices of maintaining a hostile work environment based on race, color, and/or national origin in violation of Title VII of the Civil Rights Act of 1964, as amended, 42 U.S.C. § 2000e, et. seq. ("Title VII") and subjecting individuals to disparate terms and conditions based on race, color and/or national origin. This action seeks to provide appropriate relief to a class of individuals adversely affected by such practices. Plaintiff, the U.S. Equal Employment Opportunity Commission ("EEOC"), contends Defendant, Albertson's, Inc. ("Albertson's"), has discriminated against a class of employees at the Distribution Center in Aurora, Colorado because of their race, color, and/or national origin. EEOC alleges that Albertson's subjected this class of employees to a hostile work environment and unequal terms and conditions of employment in violation of Title VII. EEOC also alleges that Albertson's failed to take prompt and effective remedial action. Jurisdiction of this court is invoked pursuant to 28 U.S.C. §§ 451, 1331, 1337, 1343 and 1345. This action is authorized and instituted pursuant to Sections 703(a), 704, 706(f)(1), 706(f)(3), of Title VII of the Civil Rights Act of 1964, as amended, 42 U.S.C. §§ 2000e-2(a), 2000e-2, 2000e-2(f)(1), 2000e-5(f)(3), and Section 102 of the Civil Rights Act of 1991; 42 U.S.C. §§ 1991a-2; The employment practices alleged to be unlawful were committed within the jurisdiction of the United States District Court, District of Colorado. PARTIES 3. Plaintiff EEOC is the agency of the United States of America charged with the administration, interpretation and enforcement of Title VII, and is expressly authorized to bring this action by Sections 706(f)(1) and 706(f)(3) of Title VII, 42 U.S.C. §§ 2000e-5(f)(1) and (3). 4. At all relevant times, Defendant has continuously been and is now doing business in the State of Colorado and has continuously had at least fifteen (15) employees. 5. At all relevant times, Defendant has continuously been and is now doing business in the State of Colorado and has continuously had at least fifteen (15) employees. 6. At all relevant times, Defendant has continuously been and is now doing business in the State of Colorado and has continuously had at least fifteen (15) employees. 7. Plaintiff EEOC is the agency of the United States of America charged with the administration, interpretation and enforcement of Title VII, and is expressly authorized to bring this action by Sections 706(f)(1) and 706(f)(3) of Title VII, 42 U.S.C. §§ 2000e-5(f)(1) and (3). 8. Since at least 1995, Defendant has engaged in unlawful employment practices at its Distribution Center in Aurora, Colorado, in violation of section 703(a) of Title VII, 42 U.S.C. § 2000e-2(a), by allowing its Distribution Center employees to be subjected to repeated and continuing racial and national origin based harassment, comments, and graffiti including, but not limited to: (a) swastikas and racial slurs on the bathroom fixtures and walls, and on the warehouse racks, including the use of the racial slurs "n-word" and "spic," including references to "bastard babies" and including drawings of Black and Hispanic individuals with ropes around their necks; (b) writing on equipment with the term "lazy n-word" and other racial and national origin based slurs; (c) statements to Mr. Ricks and other black employees, calling them "niggers" or "lazy niggers"; (d) writing of racial and national origin based slurs on the wall of the candy room with such phrases as "the only good spic is a dead spic." 9. In or about April of 2002, supervisor Jln Hayes ("Hayes") approached Mr. Ricks during break time, grabbed Mr. Ricks' papers, ordered him off the phone, was physically aggressive and stalked behind Mr. Ricks as he walked away, stating to Mr. Ricks, "There you go, acting like a n-word." 10. The offensive conduct was sufficiently severe or pervasive as to alter the terms and conditions of employees subjected to the conduct. 11. The offensive conduct in the workplace initiated and/or participated in by employees, including but not limited to managers, constitutes harassment. 12. The harassment in the work place created a hostile work environment based on race, color, and/or national origin. 13. Defendant was aware of the harassment and that some managers participated in the harassment. 14. Defendant failed to take reasonable measures to prevent and promptly correct harassment in the workplace. Defendant violated its own harassment policy by its inaction. For example: (a) swastikas and other derogatory racial and ethnic writings on walls and equipment remained continuously in place for periods of months or even years, even when management was explicitly advised of the graffiti; (b) in May 2001, Jln Hortley ("Hortley"), Operations Manager, responded to a complaint regarding the graffiti and other harassment by telling the employee that he was "parricide" and "racist" and took no further action; (c) based upon information and belief, Defendant CEO Gary Sheriff failed to respond to a letter from a black employee which included many examples of race discrimination; and (d) Human Resources failed to respond when a black employee reported threats of physical violence from a white management official. 15. Defendant tolerated the hostile work environment on a continuing basis since at least 1995. 16. The effect of the events described above, including the harassment by managers and Defendant's failure to promptly and adequately respond to employee complaints of harassment pursuant to its harassment policies and procedures, has been to deprive individuals of equal employment opportunities based on their race, color, and/or national origin. 17. The unlawful employment practices described above were intentional. 18. The unlawful employment practices described above were done with notice or with reckless indifference to the federally protected rights of employees based on their race, color, and/or national origin. 19. The effect of the events described above has been to deprive individuals of equal employment opportunities based on their race, color, and/or national origin. PRAYER FOR RELIEF WHEREFORE, the Commission respectfully requests

1,21 Top

Figure 2: Sample output from huggingface.py. Warning: Offensive Language

2.5 Documentation

Install openapi generator cli: <https://openapi-generator.tech/docs/installation/>

1. `openapi-generator-cli generate -i ch_openapi.json -g python` will generate the api. 2. specify your clearinghouse.net access token `export CLEARINGHOUSE_API_TOKEN=<YOUR API`
3. run `test.py` in the api directory. It should say you have successfully authenticated. Also see docs directory for api usage.

2.6 Results Visualization

```
(base) michael@michael-s-ml:~/GeorgiaTech/Law_Data_Design/api$ python3 test.py
The response of DefaultApi->test_get:

TestGet200Response(message='Success, you have access to the Civil Rights Clearinghouse Litigation API!')
(base) michael@michael-s-ml:~/GeorgiaTech/Law_Data_Design/api$
```

Figure 3: Enter Caption

3 Next Week's proposal

- Continue `ch_openapi.json` so we can call more than just the test endpoint
- Receive subteam assignment and subteam meeting time
- Read on T5, BART, and/or sequence to sequence
- Once github has been set up by Dr. Alexander, provide `hugging_face.py` to rest of team.

Human Augmented Analytics Group Report

Natural Language Processing - Sentencias

Week 1

Víctor C. Fernández
August 2024

CONTENTS

1 Abstracts	2
2 Scripts and Code Blocks	4
3 Documentation	9
4 Script Validation(optional)	11
5 Results Visualization	11
6 Proof of Work	14
7 Next Week's Proposal	14

1 ABSTRACTS

1. **Title:** DiffuSum: Generation Enhanced Extractive Summarization with Diffusion

• **URL:** <https://aclanthology.org/2023.findings-acl.828.pdf>

• **Abstract:** Extractive summarization aims to form a summary by directly extracting sentences from the source document. Existing works mostly formulate it as a sequence labeling problem by making individual sentence label predictions. This paper proposes DiffuSum, a novel paradigm for extractive summarization, by directly generating the desired summary sentence representations with diffusion models and extracting sentences based on sentence representation matching. In addition, DiffuSum jointly optimizes a contrastive sentence encoder with a matching loss for sentence representation alignment and a multi-class contrastive loss for representation diversity. Experimental results show that DiffuSum achieves the new state-of-the-art extractive results on CNN/DailyMail with ROUGE scores of 44.83/22.56/40.56. Experiments on the other two datasets with different summary lengths and cross-dataset evaluation also demonstrate the effectiveness of DiffuSum. The strong performance of our framework shows the great potential of adapting generative models for extractive summarization.

• **Summary:** The paper provides details on DiffuSum, a method for extractive summarization that utilizes diffusion models to generate representations of ideal summary sentences, subsequently selecting the most fitting sentences from the original document. The approach emphasizes the joint optimization of sentence encoding and diffusion generation modules to ensure both accurate representation and diversity.

• **Relevance:** The paper offers a way of extracting relevant information from a document, which could help in the extraction of key information from judges' written decisions. The capability to generate concise and informative summaries of these decisions could significantly aid in constructing a structured archive of court data.

2. **Title:** Simple Yet Powerful: An Overlooked Architecture for Nested Named

Entity Recognition

- **URL:** <https://aclanthology.org/2022.coling-1.184.pdf>
- **Abstract:** Named Entity Recognition (NER) is an important task in Natural Language Processing that aims to identify text spans belonging to predefined categories. Traditional NER systems ignore nested entities, which are entities contained in other entity mentions. Although several methods have been proposed to address this case, most of them rely on complex task-specific structures and ignore potentially useful baselines for the task. We argue that this creates an overly optimistic impression of their performance. This paper revisits the Multiple LSTM-CRF (MLC) model, a simple, overlooked, yet powerful approach based on training independent sequence labeling models for each entity type. Extensive experiments with three nested NER corpora show that, regardless of the simplicity of this model, its performance is better or at least as well as more sophisticated methods. Furthermore, we show that the MLC architecture achieves state-of-the-art results in the Chilean Waiting List corpus by including pre-trained language models. In addition, we implemented an open-source library that computes task-specific metrics for nested NER. The results suggest that metrics used in previous work do not measure well the ability of a model to detect nested entities, while our metrics provide new evidence on how existing approaches handle the task.
- **Summary:** The paper investigates a Multiple LSTM-CRF (MLC) architecture, for Nested Named Entity Recognition (NER). The MLC model trains independent sequence labeling models for each entity type, addressing the challenges of nested entities and multi-label entities. The paper also highlights the inadequacy of current evaluation metrics for nested NER and proposes new task-specific metrics.
- **Relevance:** The paper focuses on Nested Named Entity Recognition, which is directly applicable to extracting key information from legal documents. The MLC architecture's ability to handle nested entities and multi-label entities could be particularly useful in identifying complex relationships and overlapping information within judges' written decisions.

2 SCRIPTS AND CODE BLOCKS

All scripts will be uploaded to <https://github.com/Human-Augment-Analytics/NLP-Gen>, but haven't been able to do so yet as I don't have access granted for pushing code into the repo.

I've divided into different code blocks what is currently presented as different blocks in the jupyter notebook:

```
1 # Import all necessary dependencies
2 import pypdf
3
4 import nltk
5 from nltk.tokenize import word_tokenize, sent_tokenize
6 from nltk.corpus import stopwords
7 from collections import Counter
8
9 from textblob import TextBlob
10 import spacy
11
12 from gensim.models import LdaModel
13 from gensim.corpora import Dictionary
14
15 import pandas as pd
16 import altair as alt
17 import matplotlib.pyplot as plt
18
19 from wordcloud import WordCloud
20
21 import string
22 import os
23
24 nltk.download('punkt_tab')
25 nltk.download('stopwords')
26 nltk.download('averaged_perceptron_tagger_eng')
```

Listing 1—Imports

```
1 def extract_text_from_pdf(pdf_path):
2     with open(pdf_path, 'rb') as pdf_file:
3         pdf_reader = pypdf.PdfReader(pdf_file)
4         num_pages = len(pdf_reader.pages)
5         text = ""
6         for page_num in range(num_pages):
7             page = pdf_reader.pages[page_num]
```



```

8         text += page.extract_text()
9     return text

```

Listing 2—Extract Text function

```

1 def preprocess_text(text):
2     text = text.lower()
3     text = "".join([char for char in text if char not in string.punctuation])
4     text = " ".join(text.split())
5     return text

```

Listing 3—Preprocess function

```

1 def preprocess_text_with_punctuation(text):
2     text = text.lower()
3     punctuation_to_keep = '?!'
4     text = "".join([char if char in punctuation_to_keep or char not in string.
5     punctuation else ' ' for char in text])
6     text = " ".join(text.split())
7     return text

```

Listing 4—Preprocess function keeping punctuation for later separating sentences

```

1 def tokenize_text(text):
2     words = word_tokenize(text)
3     sentences = sent_tokenize(text)
4     return words, sentences

```

Listing 5—Tokenize function

```

1 def generate_word_cloud(text):
2     words = word_tokenize(text)
3     stop_words = set(stopwords.words('spanish'))
4     filtered_words = [word for word in words if word.lower() not in stop_words]
5
6     word_counts = Counter(filtered_words)
7     wordcloud = WordCloud(width=800, height=400, background_color='white').
8     generate_from_frequencies(word_counts)
9
10    plt.figure(figsize=(10, 5))
11    plt.imshow(wordcloud, interpolation='bilinear')
12    plt.axis('off')
13    plt.show()

```

Listing 6—Generate word cloud function

```

1 def combine_all_pdfs_text(pdf_folder_path):
2     # Get all pdf files in folder
3     pdf_files = [f for f in os.listdir(pdf_folder_path) if f.endswith('.pdf')]
4
5     all_text = ""
6     for pdf_file in pdf_files:
7         pdf_path = os.path.join(pdf_folder_path, pdf_file)
8         print("pdf_path", pdf_path)
9
10        # Extract and preprocess the text
11        text = extract_text_from_pdf(pdf_path)
12        all_text += preprocessed_text + " "
13
14    return all_text

```

Listing 7—Combine pdfs function

```

1 all_pdfs_text = combine_all_pdfs_text('./sentencias')
2 preprocessed_text = preprocess_text(all_pdfs_text)
3
4 # Generate and display the word cloud
5 print(f"\nWord Cloud for all pdfs:")
6 generate_word_cloud(preprocessed_text)

```

Listing 8—Additional processing

```

1 # Tokenize combined text
2 preprocessed_text_2 = preprocess_text_with_punctuation(all_pdfs_text)
3 words, _ = tokenize_text(preprocessed_text)
4 _, sentences = tokenize_text(preprocessed_text_2)

```

Listing 9—Retrieve preprocessed text

```

1 # Perform other analyses:
2 # Word Frequency
3 df_words = pd.DataFrame(Counter(words).most_common(), columns=['Word', 'Frequency'])
4
5 # Remove stop words
6 stop_words = set(stopwords.words('spanish'))
7 filtered_words = [word for word in words if word not in stop_words]
8 df_words_no_stop = pd.DataFrame(Counter(filtered_words).most_common(), columns=['Word',
9                                     'Frequency'])
10
11 chart1 = alt.Chart(df_words_no_stop.head(20)).mark_bar().encode(
12     x=alt.X('Word:N', sort='-y'),
13     y=alt.Y('Frequency:Q'),

```

```

13     tooltip = ['Word', 'Frequency']
14 ).properties(title='Top 20 Most Frequent Words (Excluding Stop Words)').interactive()
15 chart1.display()

```

Listing 10—Retrieve word frequency in text

```

1 # N-grams
2 bigrams = list(nltk.bigrams(words))
3 df_bigrams = pd.DataFrame(Counter(bigrams).most_common(), columns=['Bigram', 'Frequency'
4 ])
5 df_bigrams['Bigram'] = df_bigrams['Bigram'].astype(str)
6 chart2 = alt.Chart(df_bigrams.head(10)).mark_bar().encode(
7     x=alt.X('Bigram:N', sort='-y'),
8     y=alt.Y('Frequency:Q'),
9     tooltip = ['Bigram', 'Frequency']
10 ).properties(title='Top 10 Most Frequent Bigrams').interactive()
11 chart2.display()

```

Listing 11—Retrieve bigrams

```

1 trigrams = list(nltk.trigrams(words))
2 df_trigrams = pd.DataFrame(Counter(trigrams).most_common(), columns=['Trigram', '
3     Frequency'])
4 df_trigrams['Trigram'] = df_trigrams['Trigram'].astype(str)
5
6 chart3 = alt.Chart(df_trigrams.head(10)).mark_bar().encode(
7     x=alt.X('Trigram:N', sort='-y'),
8     y=alt.Y('Frequency:Q'),
9     tooltip = ['Trigram', 'Frequency']
10 ).properties(title='Top 10 Most Frequent Trigrams').interactive()
11 chart3.display()

```

Listing 12—Retrieve trigrams

```

1 # Sentiment Analysis
2 sentiments = [TextBlob(sentence).sentiment for sentence in sentences]
3 df_sentiment = pd.DataFrame({'Sentence': sentences, 'Polarity': [s.polarity for s in
4     sentiments], 'Subjectivity': [s.subjectivity for s in sentiments]})
5 chart4 = alt.Chart(df_sentiment).mark_bar().encode(
6     x=alt.X('Polarity:Q', bin=True),
7     y=alt.Y('count()', title='Number of Sentences'),
8     tooltip = [alt.Tooltip('Polarity:Q', bin=True), 'count()']
9 ).properties(title='Distribution of Sentiment Polarity').interactive()

```

```

10
11 chart5 = alt.Chart(df_sentiment).mark_circle().encode(
12     x='Polarity',
13     y='Subjectivity',
14     tooltip = ['Polarity', 'Subjectivity']
15 ).properties(title='Sentiment Polarity vs Subjectivity').interactive()
16
17 print(df_sentiment)
18
19 chart4.display()
20 chart5.display()

```

Listing 13—Retrieve Sentiment analysis

```

1 # Sentence Length
2 print (sentences)
3 sentence_lengths = [len(sentence.split()) for sentence in sentences]
4 df_sentence_length = pd.DataFrame({'Sentence': sentences, 'Length': sentence_lengths})
5
6 print(df_sentence_length)
7
8 chart6 = alt.Chart(df_sentence_length).mark_bar().encode(
9     x=alt.X('Length:Q', bin=True),
10    y=alt.Y('count()'), title='Number of Sentences'),
11    tooltip = [alt.Tooltip('Length:Q', bin=True), 'count()']
12 ).properties(title='Distribution of Sentence Lengths').interactive()
13 chart6.display()

```

Listing 14—Retrieve sentences lengths

```

1 # POS Tagging
2 pos_tags = nltk.pos_tag(words)
3 df_pos_tags = pd.DataFrame(Counter([tag for _, tag in pos_tags]).most_common(), columns
4     =['POS Tag', 'Frequency'])
5
6 print (df_pos_tags)
7
8 chart7 = alt.Chart(df_pos_tags).mark_bar().encode(
9     x=alt.X('POS Tag:N', sort='-y'),
10    y=alt.Y('Frequency:Q'),
11    tooltip = ['POS Tag', 'Frequency']
12 ).properties(title='Frequency of POS Tags').interactive()
13 chart7.display()

```

Listing 15—Retrieve POS tags

```

1 # NER
2 nlp = spacy.load("es_core_news_md") # Load Spanish spaCy model
3 entities = []
4 labels = []
5 for sentence in sentences:
6     doc = nlp(sentence)
7     for ent in doc.ents:
8         entities.append(ent.text)
9         labels.append(ent.label_)
10 df_ner = pd.DataFrame(Counter(zip(entities, labels)).most_common(), columns=['
    Entity_Label', 'Frequency'])
11 df_ner[['Entity', 'Label']] = pd.DataFrame(df_ner['Entity_Label'].tolist(), index=df_ner
    .index)
12 df_ner = df_ner[['Label', 'Frequency']]
13 df_ner = df_ner.groupby('Label').sum().reset_index()
14
15 entity_label_descriptions = {
16     'LOC': 'Locations such as geographical entities, buildings, airports, etc.',
17     'MISC': 'Miscellaneous entities, not belonging to any of the other categories',
18     'ORG': 'Organizations, companies, institutions, etc.',
19     'PER': 'People, including names of individuals'
20 }
21
22 df_ner['Description'] = df_ner['Label'].map(entity_label_descriptions)
23
24 print (df_ner)
25
26 chart8 = alt.Chart(df_ner).mark_bar().encode(
27     x=alt.X('Description:N', sort='-y'),
28     y=alt.Y('Frequency:Q'),
29     tooltip = ['Description', 'Frequency']
30 ).properties(title='Frequency of Entity Labels').interactive()
31
32 chart8.display()

```

Listing 16—Retrieve NER

3 DOCUMENTATION

1. Data Collection and Preprocessing:

- A set of judicial decisions (sentencias) in pdf format was obtained from Dr. Alexander, originating from the National School of the Judiciary in the Dominican Republic.

- Text was extracted from PDF files using the pypdf library.
- The extracted text was preprocessed, which included:
 - Converting all text to lowercase.
 - Removing punctuation (or keeping some specific punctuation marks for sentence segmentation).
 - Removing extra whitespace.

2. Text Analysis and Feature Extraction:

- The preprocessed text was tokenized into words and sentences using NLTK's `word_tokenize` and `sent_tokenize` functions.
- Word clouds were generated to visualize the most frequent words in the corpus.
- Word frequency analysis was performed, including:
 - Counting the frequency of individual words.
 - Removing Spanish stop words using NLTK's stopwords.
 - Identifying and visualizing the most frequent words and bigrams (two-word sequences) and trigrams (three-word sequences).
- Sentiment analysis was conducted using TextBlob to determine the polarity (positive/negative) and subjectivity of each sentence.
- Sentence length analysis was performed by counting the number of words in each sentence.
- Part-of-speech (POS) tagging was performed using NLTK to identify the grammatical roles of words in the corpus.
- Named entity recognition (NER) was performed using spacy's Spanish language model (`es_core_news_md`) to identify and categorize entities like people, organizations, and locations.

3. Visualization and Reporting:

- The results of the analyses were visualized using Altair charts and matplotlib.
- DataFrames were used to organize and manipulate the extracted data.
- Charts were generated to represent the frequency distributions of various features (word frequency, sentiment polarity, sentence length, POS tags, entity labels).

4 SCRIPT VALIDATION(OPTIONAL)

Still awaiting on the Kickoff meeting that will be carried out on Monday 26th of August. Until then, not certain that any of the data I've retrieve so far will align with the goals of the project.

5 RESULTS VISUALIZATION



Figure 1—Word cloud

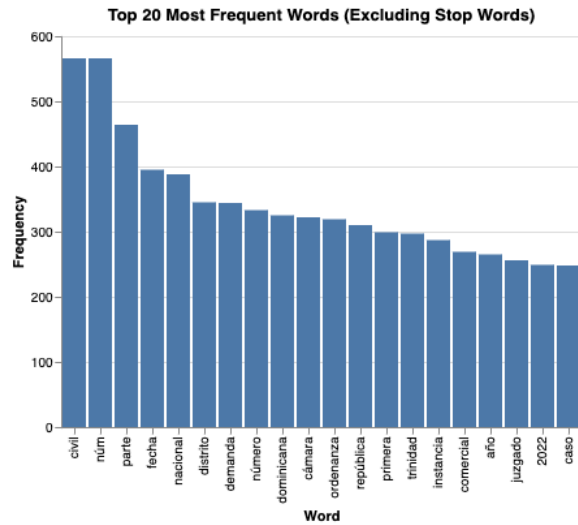


Figure 2—Top 20 most frequent words

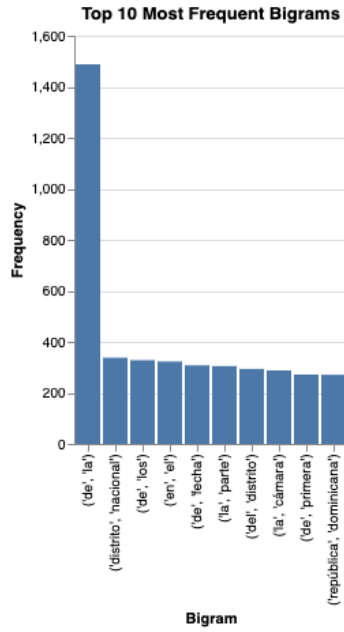


Figure 3—Top 10 most frequent bigrams

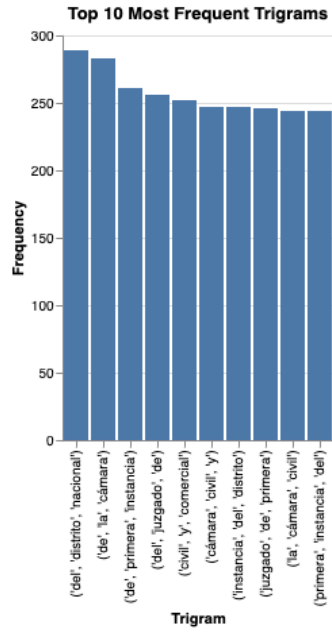


Figure 4—Top 10 most frequent trigrams

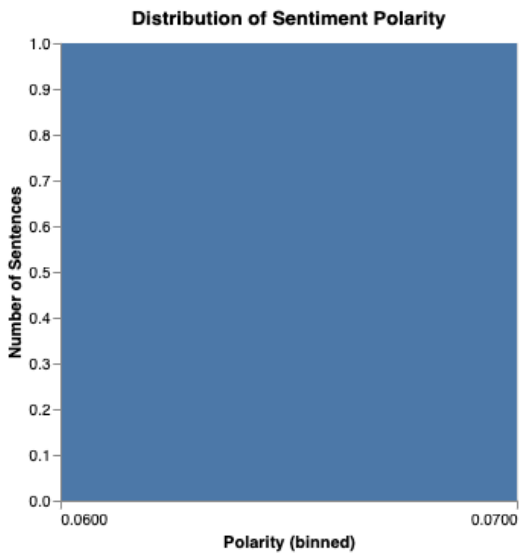


Figure 5—Sentiment polarity

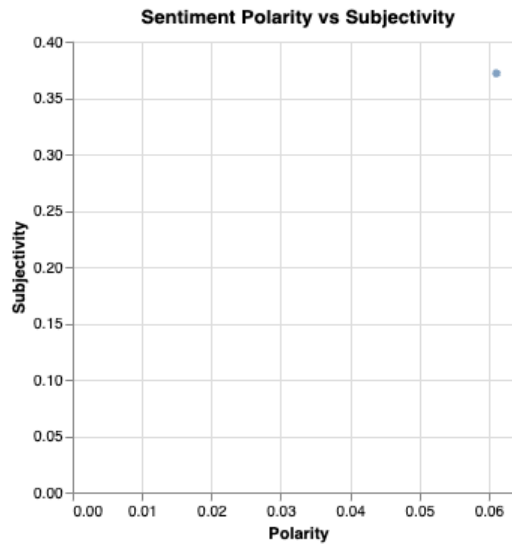


Figure 6—Sentiment polarity vs subjectivity

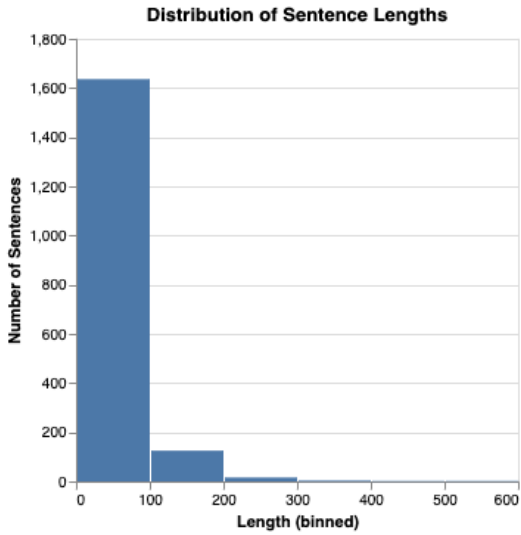


Figure 7—Distribution of sentence lengths

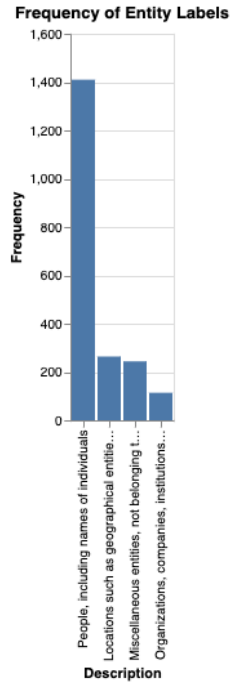


Figure 8—Entity labels frequency

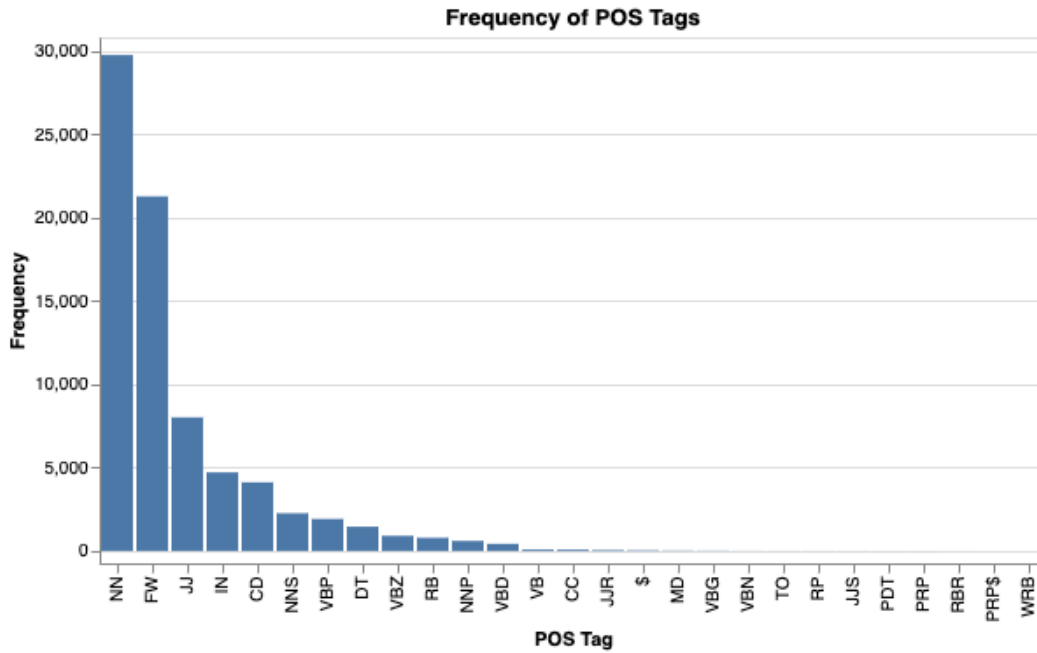


Figure 9—POS tags frequency

6 PROOF OF WORK

The results obtained seem to be reliable for a first data pass. It's important to highlight that text preprocessing is fairly simple at this point and input text will definitely need additional filtering to rule out tokens lacking value such as simple nouns like the word "number" in Spanish, or similar content that may lack meaning in terms of interpreting the content of the document. Besides that, sentiment analysis seems neutral which makes sense given the nature of the documents and words with the highest frequency seem to be simple nouns and generic adjectives related to the type of document.

In general terms, a simple data ingestion mechanism was created that can now allow for further investigation if the data source happens to be the provided pdf documents.

Will keep investigating more in the following week based on the outcome of the kickoff meeting with Dr. Alexander.

7 NEXT WEEK'S PROPOSAL

- Meet with Dr. Alexander for the project kick-off and gain insights on what our goals and expectations should be on Monday.
- Join the NLP group weekly meeting on Monday.
- Gain insights on data ingestion for the NLP-Setencias documents.
- Gain knowledge on the source documents and their context information.
- Further develop data ingestion and data processing pipelines for the pdf documents.
- Look for models trained with legal Spanish language.

HAAG - NLP | Fall 2024

Alejandro Gomez

August 23, 2024

1 Abstract

Three NLP (Natural Language Processing) automated summarization techniques were tested on a special collection of Catholic Pamphlets acquired by Hesburgh Libraries. The automated summaries were generated after feeding the pamphlets as .pdf files into an OCR pipeline. Extensive data cleaning and text preprocessing were necessary before the computer summarization algorithms could be launched. Using the standard ROUGE F1 scoring technique, the Bert Extractive Summarizer technique had the best summarization score. It most closely matched the human reference summaries. The BERT Extractive technique yielded an average Rouge F1 score of 0.239. The Gensim python package implementation of TextRank scored at .151. A hand-implemented TextRank algorithm created summaries that scored at 0.144. This article covers the implementation of automated pipelines to read PDF text, the strengths and weakness of automated summarization techniques, and what the successes and failures of these summaries mean for their potential to be used in Hesburgh Libraries. [doi link\[F1a07\]](#)

2 Scripts and Code Blocks

2.1 Scripts and Code Blocks Info

```
1 import numpy as np
2 pdf_titles_by_group = np.array([
3     ["title_1", "title_2"],
4     ["title_3", "title_4"]
5 ])
6 print(pdf_titles_by_group.shape)
7 print(pdf_titles_by_group.ndim)
8 print(pdf_titles_by_group.size)
```

Listing 1: numpy example

```
1 import pandas as pd
2 df = pd.DataFrame(
3     {
4         "Group 1" : [
5             "title_1", "title_2"
6         ],
7         "Group 2" : [
8             "title_3", "title_4"
9         ]
10    }
11 )
12 print(f" Group 2: {df['Group 2']}")
```

Listing 2: pandas example

```
1 import matplotlib.pyplot as plt
2
3 pdf_titles = ["title_1", "title_2", "title_3", "title_4"]
4 values = [34, 23, 54,7]
5 plt.figure(1)
6 plt.suptitle('Research Papers')
7 plt.bar(pdf_titles, values)
8 plt.show()
```

Listing 3: matplotlib example

2.2 Documentation

The NLP team's first meeting with Dr. Alexander will be tomorrow 7amPT/10amET. We expect to get a project synopsis at that time and discuss first steps for preliminary research and design so that we can begin project design and coding. Due to these circumstances, I've taken this time to familiarize myself with setting up an environment for scientific Python coding with WSL2 + conda environments + nump/pandas/matplotlib. [Scripts in GitHub Repo](#)

2.3 Script Validation (optional)

NA

2.4 Results Visualization

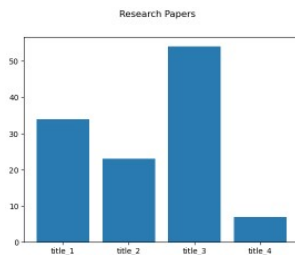


Figure 1: matplotlib plot

2.5 Proof of Work

This was a simple exercise. In the future, results will be evaluated noting oddities and reliability.

3 Next Week's Proposal

- Attend NLP student members meeting
- Discuss a plan for our project objective
- Begin working on a design and architecture
- Familiarize with tools

References

- [Fla07] Jeremiah Flannery. Using nlp to generate marc summary fields for notre dame 's catholic pamphlets. *International journal of librarianship*, 5(1):20–35, 2020-07.

HAAG NLP DR — Week 1 Report

Karol Gutierrez

August 23, 2024

1 Literature Review

Paper: Generating Wikipedia by Summarizing Long Sequences [LSP⁺18].

1.1 Abstract

We show that generating English Wikipedia articles can be approached as a multi- document summarization of source documents. We use extractive summarization to coarsely identify salient information and a neural abstractive model to generate the article. For the abstractive model, we introduce a decoder-only architecture that can scalably attend to very long sequences, much longer than typical encoder- decoder architectures used in sequence transduction. We show that this model can generate fluent, coherent multi-sentence paragraphs and even whole Wikipedia articles. When given reference documents, we show it can extract relevant factual information as reflected in perplexity, ROUGE scores and human evaluations.

1.2 Summary

The text discusses a research project by Google Brain that focuses on generating English Wikipedia articles through a multi-document summarization approach. They use extractive summarization to identify key information and a neural abstractive model to generate the articles, introducing a decoder-only architecture that can handle long sequences efficiently. The study demonstrates the effectiveness of their model in extracting factual information from reference documents, as evaluated through perplexity, ROUGE scores, and human assessments.

2 Scripts and code blocks

All the existing code exists in the following [repository](#).

This is a code block that calls the ‘summarizer‘ function and then print the texts and generate the plots. It uses sample data extracted from Wikipedia articles as well as the abstract from the paper used in the Literature Review.

The current code logic is explained in the following diagram.

3 Documentation

The documentation is also present in the README.md file in the [repository](#).

3.1 README.md

```
# nlp-dr  
Code snippets for NLP project
```

```
then install torch manually, such as  
conda install pytorch torchvision -c pytorch
```

```

39 summaries = [summarizer(text, max_length=50, min_length=25, do_sample=False)[0]['summary_text'] for text in texts]
40
41 # Calculate lengths of original vs summaries texts
42 original_lengths = [len(text.split()) for text in texts]
43 summary_lengths = [len(summary.split()) for summary in summaries]
44
45 for i in range(len(texts)):
46     print(f"Original text {i+1}:", texts[i])
47     print(f"Summarized text {i+1}:", summaries[i])
48     print()
49
50
51 # Visualize the length of the original text vs the summaries
52 labels = ['Text 1', 'Text 2', 'Text 3', 'Text 4', 'Text 5']
53 x = range(len(labels))
54
55 plt.figure(figsize=(10, 6))
56 plt.bar(x, original_lengths, width=0.4, label='Original Text', align='center')
57 plt.bar(x, summary_lengths, width=0.4, label='Summary', align='edge')
58
59 plt.xticks(x, labels)
60 plt.ylabel('Word Count')
61 plt.title('Original Text vs Summary Length')
62 plt.legend()
63 plt.show()

```

Figure 1: Code block.

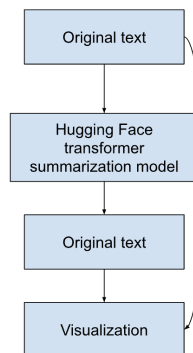


Figure 2: Code logic flow chart.

NLP-DR: Text summarization

Sample script to summarize text using the Hugging Face ‘transformers’ library. The script takes text fragments, generates summaries and generate plots.

Project Structure

- **summarization_example.py**: The main script that performs text summarization and plots.

Setup Instructions

Prerequisites

- **Python 3.9** or later.
- **Miniconda or Anaconda** installed on your system.

Setup the Environment

1. Run ‘conda env create -f environment.yaml’
2. Activate environment using ‘conda activate haag-nlp’
3. Manually install pytorch, use reference from ‘<https://pytorch.org/get-started/locally/>’. For example: ‘conda install pytorch torchvision -c pytorch’

```
### Run the code
```

```
Use 'python summarization_example.py'.
```

4 Script Validation

It doesn't apply at this point of the development of the project.

5 Results Visualization

After doing the setup steps, the sample code should generate the following plots and also print the original and summarized versions of the texts.

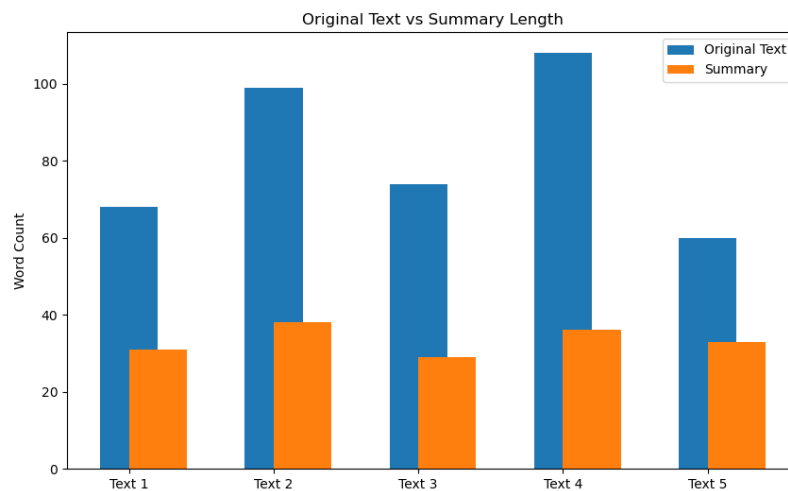


Figure 3: Comparison of text lengths.

6 Proof of Work

These results are demonstrations of existing libraries for NLP. The only available proof of work is the code running successfully after doing the setup steps from the documentation.

7 Next Week's Proposal

- Sync with Dr. Alexander and team to get more details on the requirements for the project and general overview of tasks.
- Fulfill my role as Meet Manager/Documentor by working on the tasks expected for my position.
- Further literature review of techniques that could be applied to this project.
- Work using the dataset from the project, initial tasks could be related to data extraction.

References

[LSP⁺18] Peter J. Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. Generating wikipedia by summarizing long sequences, 2018.


```
(haag-nlp) * nlp-dr git:(main) *
(haag-nlp) * nlp-dr git:(main) * python summarization_example.py
Original text 1:
Natural language processing (NLP) is an interdisciplinary subfield of computer science and artificial intelligence. It is primarily concerned with providing computers with the ability to process data encoded in natural language and is thus closely related to information retrieval, knowledge representation and computational linguistics, a subfield of linguistics. Typically data is collected in text corpora, using either rule-based, statistical or neural-based approaches in machine learning and deep learning.
Summarized text 1: Natural language processing (NLP) is an interdisciplinary subfield of computer science and artificial intelligence. It is primarily concerned with providing computers with the ability to process data encoded in natural language.
Original text 2:
We show that generating English Wikipedia articles can be approached as a multi-document summarization of source documents. We use extractive summarization to coarsely identify salient information and a neural abstractive model to generate the article. For the abstractive model, we introduce a decoder-only architecture that can scalably attend to very long sequences, much longer than typical encoder-decoder architectures used in sequence transduction. We show that this model can generate fluent, coherent multi-sentence paragraphs and even whole Wikipedia articles. When given reference documents, we show it can extract relevant factual information as reflected in perplexity, ROUGE scores, and human evaluations.
Summarized text 2: We show that generating English Wikipedia articles can be approached as a multi-document summarization of source documents. We use extractive summarization to coarsely identify salient information and a neural abstractive model to generate the article. When given reference
Original text 3:
Portrait of Ambroise Vollard is an 1899 oil-on-canvas portrait by Paul Cézanne of his art dealer Ambroise Vollard. It was bequeathed by Vollard on his death to the Petit Palais in Paris, where it is still housed today. Like many of his portraits, the Portrait of Ambroise Vollard displays the significant role of the subject in Cézanne's life, and specifically, the artist's gratitude for promoting his work and establishing his reputation as an artist.
Summarized text 3: Portrait of Ambroise Vollard is an 1899 oil-on-canvas portrait by Paul Cézanne of his art dealer. It was bequeathed on his death to the Petit Palais in Paris, where
Original text 4:
The Dominican Republic is a North American country on the island of Hispaniola in the Greater Antilles archipelago of the Caribbean Sea, bordered by the Atlantic Ocean to the north. It occupies the eastern five-eighths of the island, which it shares with Haiti, making Hispaniola one of only two Caribbean islands, along with Saint Martin, that is shared by two sovereign states. It is the second-largest nation in the Antilles by area (after Cuba) at 48,671 square kilometers (18,792 sq mi), and second-largest by population, with approximately 11.4 million people in 2024, of whom approximately 3.6 million live in the metropolitan area of Santo Domingo, the capital city.
Summarized text 4: The Dominican Republic is a North American country on the island of Hispaniola. It occupies the eastern five-eighths of the island, which it shares with Haiti. It is the second-largest nation in the Antilles by area
Original text 5:
Artificial intelligence (AI), in its broadest sense, is intelligence exhibited by machines, particularly computer systems. It is a field of research in computer science that develops and studies methods and software that enable machines to perceive their environment and use learning and intelligence to take actions that maximize their chances of achieving defined goals.[1] Such machines may be called AIs.
Summarized text 5: Artificial intelligence (AI) is intelligence exhibited by machines, particularly computer systems. Such machines may be called AIs. It is a field of research in computer science that develops and studies methods and software.
```

Figure 4: Proof of code working.

Week 1 - Research Report

Thomas Orth

August 2024

Contents

1 Completed work	1
2 Abstracts	2
3 Relevant Info	3
4 Scripts	3
5 Documentation	6
6 Results	7
6.1 OCR / Parsing Results	7
6.2 Summary Result	8
7 Proof of Results	8
7.1 OCR	8
7.1.1 Documentation	8
7.1.2 Oddities	8
7.1.3 Open Questions / Routes to take	8
7.2 Summary Model results	8
7.2.1 Documentation	8
7.2.2 Known Limitations	9
7.2.3 Oddities	9
7.2.4 Open Questions / Routes to take	9
8 Next Steps	9

1 Completed work

- Met with new NLP members. Held meeting to go over course logistics
- Attended kickoff event held by Dr. Alexander in collaboration with her VIP class

- Experimented with unstructured, a python package for pulling out data from PDFs, to see its effectiveness in OCRing
- Experimented with a pre-trained long document model for summarization to gauge zero-shot performance

2 Abstracts

- Title: Abstractive Summarization of Dutch Court Verdicts Using Sequence-to-sequence Models. Conference: ACL, Proceedings of the Natural Legal Language Processing Workshop 2022.
- Abstract: With the legal sector embracing digitization, the increasing availability of information has led to a need for systems that can automatically summarize legal documents. Most existing research on legal text summarization has so far focused on extractive models, which can result in awkward summaries, as sentences in legal documents can be very long and detailed. In this study, we apply two abstractive summarization models on a Dutch legal domain dataset. The results show that existing models transfer quite well across domains and languages: the ROUGE scores of our experiments are comparable to state-of-the-art studies on English news article texts. Examining one of the models showed the capability of rewriting long legal sentences to much shorter ones, using mostly vocabulary from the source document. Human evaluation shows that for both models hand-made summaries are still perceived as more relevant and readable, and automatic summaries do not always capture elements such as background, considerations and judgement. Still, generated summaries are valuable if only a keyword summary or no summary at all is present.
- Summary: This paper explores using advanced technology to automatically summarize legal documents, finding that while the technology can create shorter summaries, human-made summaries are still considered more relevant and readable. The study shows that existing models can work well across different languages and domains, but may not capture all the important elements found in legal texts. Future work included expanding post processing to help mitigate issues like summaries that expand on unimportant details, and a more detailed analysis.
- Relevance: This closely matches the task of summarizing the clearinghouse documents. Sequence-to-sequence models are staple among complex NLP tasks. This paper helps illuminate some key ones.

3 Relevant Info

- LED model is a proposed transformer technique from 2020 for the use in long document tasks in NLP. The paper on it can be found [here](#).
- Legal documents are very long, making LED a good initial model to explore
- One source of data that concerns the NLP-Summarization project is data from the Clearinghouse as well as COVID data, but I have not explored that website much.
- In summarization, there is abstractive and extractive summarization. **Abstractive** is generating a summarization from scratch with keeping the main ideas of the text in the generated summary. **Extractive** is taking text from the given document to generate a document. The model I was exploring is **abstractive**. Source for background
- UnstructurdIO is a library for working with different documents of an unstructured nature such as PDFs. It leverages OSS tools to be able to parse PDFs, Docx files etc. It was popularized by Retrieval Augmented Generation (RAG). Source for initial code exploration

4 Scripts

1. All scripts uploaded to <https://github.com/Human-Augment-Analytics/NLP-Gen>
2. Scripts were run with the following file for testing: <https://clearinghouse-umich-production.s3.amazonaws.com/media/doc/1865.pdf>
3. Thomas-Orth/downloader.py
 - Brief Description: This is a generic download file utility to make it easier to pull PDFs down from the clearinghouse website.
 - Status: Tested via downloading of a PDF from a ClearingHouse direct link (not by their API).
 - Important Code Blocks:
 - (a) There is only 1 class in this file: Downloader. This performs the download operation for the file.
 - Screenshot of code:

```

from urllib.request import urlretrieve
from pathlib import Path
class Downloader:

    def __init__(self, url: str):
        self.url = url

    def download(self, path: str, name: str) -> Path:
        file_path = Path(path) / name
        urlretrieve(self.url, str(file_path))
        return file_path

```

Figure 1: Script for Downloader Class

4. Thomas-Orth/download_and_parse_pdf.py

- Brief Description: This script will take in 1) A pdf URL 2) The path to save that PDF to and 3) The filename for the downloaded file and then parse using UnstructuredIO. The printed and combined text is limited to Narrative Text as the body content.
- Status: Tested against example PDF with visual confirmation of the text appearing in the document.
- Important code blocks:
 - (a) First code block: Argv inputs to get the 3 parameters described in the description
 - (b) Second code block: Run unstructured on the data
 - (c) Third code block: Combine Narrative Text and print
- Screenshot:

```

import sys
from unstructured.partition.pdf import partition_pdf

from downloader import Downloader

pdf = sys.argv[1]
download_path = sys.argv[2]
file_name = sys.argv[3]

file_pdf = Downloader(pdf).download(download_path, file_name)

text = ""

content = partition_pdf(str(file_pdf))
for entry in content:
    # Possible options: {'ListItem', 'UncategorizedText', 'Title', 'NarrativeText'}
    if entry.category == "NarrativeText":
        text += entry.text

print(text)

```

Figure 2: Parsing script for PDF extraction

5. Thomas-Orth/download_parse_and_summarize.py

- Brief Description: This script takes the download and parse aspects of the Thomas-Orth/download_and_parse_pdf.py and adds a summarizer model.
- Model Chosen: This model is a pretrained LED model on legal documents that scales to 16384 input tokens.
- Status: Ran on the example data and was able to produce a summary
- Important codeblocks:
 - (a) First code block: Argv inputs to get the 3 parameters described in the description
 - (b) Second code block: Run unstructured on the data
 - (c) Third code block: Combine Narrative Text and feed into summarizer model. Print for inspection
- Screenshot:

```
import sys
from unstructured.partition.pdf import partition_pdf

from downloader import Downloader

pdf = sys.argv[1]
download_path = sys.argv[2]
file_name = sys.argv[3]

file_pdf = Downloader(pdf).download(download_path, file_name)

text = ""

content = partition_pdf(str(file_pdf))
for entry in content:
    # Possible options: {'ListItem', 'UncategorizedText', 'Title', 'NarrativeText'}
    if entry.category == "NarrativeText":
        text += entry.text

from transformers import AutoTokenizer, AutoModelForSeq2SeqLM

tokenizer = AutoTokenizer.from_pretrained("nsi319/legal-led-base-16384")
model = AutoModelForSeq2SeqLM.from_pretrained("nsi319/legal-led-base-16384").to("mps")

padding = "max_length"

input_tokenized = tokenizer.encode(text, return_tensors='pt', padding=padding, pad_to_max_length=True, truncation=True).to("mps")
summary_ids = model.generate(input_tokenized,
                             num_beams=4,
                             no_repeat_ngram_size=3,
                             length_penalty=2,
                             min_length=350,
                             max_length=1000)

summary = [tokenizer.decode(g, skip_special_tokens=True, clean_up_tokenization_spaces=False) for g in summary_ids[0]]
last_sentence = summary.rfind(".")
summary = summary[:last_sentence+1]
print(summary)
```

Figure 3: Summarization script

6. Flow Diagram for Parse and Summarization scripts:

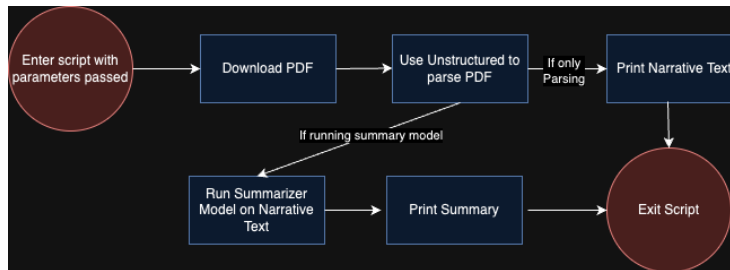


Figure 4: Flow diagram

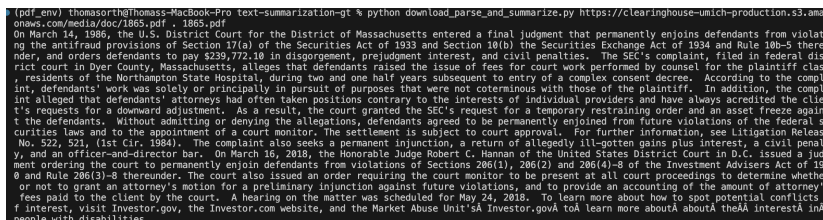
7. Running scripts:

- (a) Download the script of your choice (either `download_parse_and_summarize.py` or `download_and_parse_pdf.py`) and the `requirements.txt`
- (b) Run: `python -m pip install requirements.txt`
- (c) Run: `python <chosen script> https://clearinghouse-umich-production.s3.amazonaws.com/media/doc/1865.pdf . 1865.pdf`

5 Documentation

1. Download PDF(s) of interest
2. Perform OCR or Parsing on PDF and Extract Narrative Text
3. Feed extracted text into Pre-trained LED model
4. Output summary

6.2 Summary Result



```
(pdf env) thomasorth@thomass-MacBook-Pro text-summarization-gt % python download_parse_and_summarize.py https://clearinghouse-unich-production.s3.amazonaws.com/media/doc/1885.pdf - 1885.pdf
On March 14, 1986, the U.S. District Court for the District of Massachusetts entered a final judgment that permanently enjoins defendants from violating the antifraud provisions of Section 17(a) of the Securities Act of 1933 and Section 10(b) the Securities Exchange Act of 1934 and Rule 10b-5 thereunder, and orders defendants to pay $239,772.10 in disgorgement, prejudgment interest, and civil penalties. The SEC's complaint, filed in federal district court in Dyer County, Massachusetts, alleges that defendants raised the issue of fees for court work performed by counsel for the plaintiff class, residents of the Northampton State Hospital, during two and one half years subsequent to entry of a complex consent decree. According to the complaint, defendants' work was solely or principally in pursuit of purposes that were not coterminous with those of the plaintiff. In addition, the complaint alleged that defendants' attorneys had often taken positions contrary to the interests of individual providers and have always accredited the client's requests for a downward adjustment. As a result, the court granted the SEC's request for a temporary restraining order and an asset freeze against the defendants. Without admitting or denying the allegations, defendants agreed to be permanently enjoined from future violations of the Federal securities laws and to the appointment of a court monitor. The settlement is subject to court approval. For further information, see Litigation Release No. 522, 521, (1st Cir. 1984). The complaint also seeks a permanent injunction, a return of allegedly ill-gotten gains plus interest, a civil penalty, and an officer-and-director bar. On March 16, 2018, the Honorable Judge Robert C. Hannan of the United States District Court in D.C. issued a judgment ordering the court to permanently enjoin defendants from violations of Sections 206(1), 206(2) and 206(4)-8 of the Investment Advisers Act of 1940 and Rule 206(3)-8 thereunder. The court also issued an order requiring the court monitor to be present at all court proceedings to determine whether or not to grant an attorney's motion for a preliminary injunction against future violations, and to provide an accounting of the amount of attorney's fees paid to the client by the court. A hearing on the matter was scheduled for May 24, 2018. To learn more about how to spot potential conflicts of interest, visit Investor.gov, the Investor.com website, and the Market Abuse Unit's Investor.gov to learn more about the interest in people with disabilities.
```

Figure 7: Summary

This is the output of the LED model pretrained on legal data in Figure 7.

7 Proof of Results

7.1 OCR

7.1.1 Documentation

These results are reasonable due to the fact OCRing is not a perfect process. Unstructured has become popular amongst developers in the LLM space for its ease of use. Example of a usage of it for LLMs can be found here.

It leverages the well-known OCR models for parsing such as tesseract and PaddleOCR.

7.1.2 Oddities

So not all of the text was parsed correctly, as we saw with Figure 6. OCR isn't perfect so a few things that require either manual correction or guardrails to postprocess makes sense.

7.1.3 Open Questions / Routes to take

So this was only with Narrative Text, which is a category unstructured uses to denote main body text from what I can tell. There is also: UncategorizedText, ListItem, and Title. I do not know yet the impact of including this text would have on summarization. That is something to potentially look into.

7.2 Summary Model results

7.2.1 Documentation

The chosen LED model was taken from huggingface here. I chose LED because it is good for long documents. Additionally, it was noted in this paper that LED was pretty good on legal documents vs. Models like Pegasus. I went with

a pretrained model that was pretrained on legal documents in order to ensure it had proper legal knowledge for this zero-shot experiment.

7.2.2 Known Limitations

While this is good on long documents, it is limited to 16384 input tokens. I believe it truncates the rest of the input if it gets more than that.

7.2.3 Oddities

It seemed to have gotten the legal verbage right but one oddity was the year. It seemed to think the document was talking about 2018 when the document was talking about a case from the 1980s. I believe this is due to the training data used for the model pretraining. It probably was using more recent documents and hallucinated the year. So when we get to training, we'll want to consider the tradeoffs of training from scratch vs. pretrained on legal text.

7.2.4 Open Questions / Routes to take

So one major thing to look at is how to overcome the limited input size of models. Langchain mentions the practice of Map-reduce for summarizing found here. This is a way to chunk up text, summarize the chunks and then combine. I do not know if that works for LED or the more "modern" LLMs but such approaches should be considered as we go.

8 Next Steps

1. Perform extensive review of techniques, tools, and models that are used for summarization, both generic and specific to legal documents
2. Coordinate with selected sub-team for Dr. Alexander's VIP course and refine further tasking with sub-team members
3. Experiment further with unstructured for the different text types and PDFs
4. Start understanding the Clearinghouse API