

Automatic Text Summarization

Yuri Ho

Submitted for the Degree of Master of Science in

Data Science and Analytics



Department of Computer Science
Royal Holloway University of London
Egham, Surrey TW20 0EX, UK

August 31, 2020

Declaration

This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count: 14,878 Words

Student Name: Yuri Ho

Date of Submission: 31 August 2020

Signature: *Yuri Ho*

Abstract

Summarizing long documents is a natural way to condense information that is more palatable to readers. In recent years, there has been an explosion in the amount of text data from numerous sources. With the surplus of text data, a need has been created to automatically summarize all of this new information. For instance, search engines all employ automatic summarization techniques to present users with information related to their queries. Therefore, the importance of automatic text summarization techniques in the modern world cannot be understated.

In this project, we work with a dataset provided by the Consumer Financial Bureau of Protection. As the entries in this dataset are quite long, we want to create a summarization method that can condense this information in a way that is more presentable. To this end, we propose a novel way of automatically summarizing texts, and show that the results obtained compare favorably to widely accepted benchmarks.

We choose to focus on extractive, unsupervised methods of summarization, with emphasis on two methods: Latent Semantic Analysis and SumBasic. Our final summarizer will build upon elements from both of these algorithms. We will show that our summarizer will in fact outperform LSA and SumBasic, as well as many other common baselines.

In addition to introducing a novel way of summarization, this project aims to provide a comprehensive overview of the automatic text summarization. As this field is quite extensive, we cover all of the basics. Researchers looking to learn more about the text summarization field in general can find a lot of value in reading the Background Research section of this document.

Contents

1	Introduction	1
2	Background Research	2
2.1	Introducing Text Summarization	2
2.1.1	Extractive vs. Abstractive	2
2.1.1	Single vs. Multi-Document Summarization	2
2.1.3	Generic vs. Query-Based Summarization	2
2.1.4	The Origin of Automatic Summarization	3
2.1.5	SumBasic	4
2.1.6	Graph-based Summarizers	4
2.1.6.1	LexRank and TextRank	6
2.1.7	Topic-based Summarizers	6
2.1.7.1	Latent Semantic Analysis	6
2.1.7.1.1	Input Matrix Creation	7
2.1.7.1.2	Singular Value Decomposition	8
2.1.7.1.3	Sentence Selection	10
2.1.8	Machine Learning Summarizers	12
2.1.9	Automatic Summarization: Closing Statements	13
2.2	Evaluation Metrics: Rouge	13
2.2.1	Rouge-N	14
2.2.2	Rouge-L	15
3	Proposing a New Summarizer	16
3.1	Data	16
3.2	Baseline Selection	17
3.3	Methodology of New Summarizer	19
3.3.1	Modifying LSA	19
3.3.2	Modifying SumBasic	20
3.3.3	Combining LSAPlus and SumPlus	20
3.3.4	Reapplying LSA	22
4	Evaluation	23
5	Conclusions and Future Steps	29
6	Self-Assessment	29
6.1	Strengths	29
6.2	Weaknesses	30
7	Professional Issues	30
7.1	Transparency	31
7.2	Confirmation Bias	32
8	How to Use My Project	35

List of Tables

1	Example Sentence Vectors.....	5
2	Pairwise Cosine Similarity	5
3	Example Term-Frequency Matrix	7
4	SVD A-Matrix.....	8
5	SVD D-Matrix.....	9
6	SVD U-Matrix.....	9
7	SVD V^T -Matrix	10
8	V^T -Matrix Example for Sentence Extraction	10
9	D-Matrix for Steinberger and Jezek	11
10	V^T -Matrix for Steinberger and Jezek	11
11	Example Entry of Data.....	17
12	Baseline Rouge Scores.....	18
13	V^T -Matrix for LSA-Plus.....	20
14	Comparing Rouge-1 for LSA-Pro Against Baselines.....	23
15	Rouge-1 Scores for Various Summarizers by Text Length.....	24
16	Comparing Rouge-1 for Short Summaries.....	24
17	Comparing Rouge-1 for Medium-Length Summaries.....	25
18	Comparing Rouge-1 for Long Summaries.....	25
19	Comparing Rouge-1 for Super Long Summaries	25
20	Comparing Rouge Scores for Final Summarizer	26
21	Comparing Rouge Scores for Final Summarizer by Text Length.....	26
22	Comparing Rouge-1 Scores for Final Summarizer by Various Data Splits	27
23	Cherry picking Rouge Scores.....	30
24	Rouge-Scores of Entire Dataset	31
25	Rouge-1 Scores for Long Summaries.....	32
26	Rouge-1 Scores for Very Long Summaries	32
27	Rouge-1 Scores for Short Summaries.....	32
28	Rouge-1 Scores for Medium-length Summaries	32

List of Figures

1	Model Pipeline	19
2	Model Recap	21
3	Model Breakdown	22

List of Formulas

1	Word Probability	3
2	Term Frequency- Inverse Document Frequency	3
3	Average Word Probability	4
4	SumBasic Probability Update	4
5	Cosine Similarity.....	5
6	Max Term Frequency Normalization	8
7	LSA Sentence Score	7
8	Recall	13
9	Precision.....	13
10	F1-Rouge	14

1 Introduction

Automatic text summarization is a field that is growingly increasingly important with the abundance of data on the internet. With the overabundance of information, companies are constantly finding new ways to compress this information that is more manageable to readers. Search engines like Google, for instance, already do this: they try to extract snippets of text from webpages to help summarize important content.

Because of the necessity of summarizing information, it is no surprise that there already exist numerous popular summarizers. In this project, we will build upon an existing summarizer and test it on a dataset of interest. The dataset of interest is provided by the Consumer Financial Protection Bureau (CFPB). The CFPB is an agency of the United States government that is responsible for consumer protection of the financial sector. As defined by the US government, the CFPB writes and enforces rules governing financial institutions, examines both bank and non-bank institutions, monitors and reports on markets, and collects and tracks consumer complaints.

This dataset contains tons of financial-related complaints about issues customers are facing. CFPB regulators have to sift through hundreds of thousands of complaints before they can make an accurate assessment of the text, and accordingly make a decision about what action to take after. For instance, they need to deliberate whether or not a complaint is worth submitting to the financial institution in question. By building a summarizer that can specifically summarize these texts for regulators, it helps facilitate their jobs by reducing the amount of time they have to spend reading each customer complaint.

Therefore, the goals of this project are to:

- Provide a comprehensive overview of automatic text summarization
- Propose an innovative summarizer that can reduce overall document length to facilitate CFPB regulator tasks
- Evaluate and deliberate the efficacy of the final summarizer

I am confident that this project will set the stage for my data science journey. This project covers extremely relevant concepts, and it reinforced my understanding of how the current state of machine learning came about. Especially since my project required an enormous of research, I was able to garner a very comprehensive background about the natural language processing field in general. These included unsupervised models, supervised models, statistical models, probabilistic models, and even neural-based attention models. These are all essential ML concepts, especially the neural models. The neural network subfield in particular is undergoing a resurgence in both the academic and industrial community because of the GPU chip, which has immensely increased the computational power of today's computers.

Lastly, this project has also taught me a lot about presentation. I learned how to communicate a complicated process to a reader in an organized and clear way. More importantly, I learned the importance citing all my sources to both verify and add legitimacy to my own research. I am confident that my research will be helpful for someone who is doing research in the same field. If not the model itself that is helpful, I know that the information I compiled in this report provides a very holistic introduction into the world of automatic summarization. It will undoubtedly help guide future researchers to the relevant resources in order conduct their own studies.

2 Background Research

Natural language processing is a subfield of data science that is concerned with the analysis and processing of textual data. NLP has various applications in today's world, and one of the most popular instances of NLP applications are in automatic text summarization. Automatic text summarization is the process of shortening a set of data computationally, to create a subset that represents the most important or relevant information within the original content.

Before we dive into specifics, it would be a good idea to first highlight what this section is about. In part 2.1 we introduce the field of automatic summarization. Because of our project will focus on extractive summarization techniques, in this subsection we will focus on describing the most significant methodologies and techniques for extractive summarizers. First we will discuss unsupervised approaches, with significant time devoted to frequency-based and graph-based methods. Then, we will transition our discussion to machine learning approaches, introducing some simple techniques and then devoting more time to talk about extractive state-of-the-art methods. Lastly, we will touch upon some of the best extractive and abstractive methods, giving a high-level overview of how these powerful methods work.

Subsection 2.2 will be more brief but equally important. Here we will introduce summary evaluation techniques and discuss how to discern the effectiveness of a summarizer based on numerical results.

2.1 Introducing Text Summarization

2.1.1 Extractive vs Abstractive Summarization

Summarization techniques generally lie in two broad categories:

- **Abstractive Summarization:** The process of using new words and sentences to produce a summary. Summaries generated by humans are also called abstractive summaries.
- **Extractive Summarization:** The process of using sentences or phrases from the text itself to create a summary. This is generally done by first determining the most important sentences or phrases in a document, and then taking the most highly ranked sentences or phrases to produce a summary.

2.1.2 Single – Document vs. Multi-Document

Automatic summarization methods also differ on the amount of sources used to produce summaries. In single-document summarization, one single document is referenced, and one single summary is produced in that document. In multi-document summarization, multiple documents, all related to the same topic are analyzed together to produce a summary that stands in for all of the documents. Early work in automatic summarization dealt with single-document summarizers but with the rise of the world-wide-web and growing access to data, multi-document summarizers today are being increasingly popular [21]. Even though we will be focusing on single-document summarization, we will still use a few multi-document summarizers for this project.

2.1.3 Generic-Summaries vs. Query-Based Summaries

Generic summaries are summaries that make no assumptions about the summary produced. It simply looks at the text of a document and then produces summary. In this way, it is "generic" in the sense

that these kinds of summarizers can easily be used on all types of texts. Query-based summaries on the other hand, will produce a summary that contains information most related to a user query [21]. The user query is actually assigned a weight equal to that of the document. Thus, query-based methods take both the user-query and the document into account in order to produce a summary of interest.

2.1.4 The Origin of Automatic Summarization

While understanding the overall concept of an automatic summarizer is simple, actually implementing a summarizer is far more difficult task. Indeed, a human can easily summarize a given text, but how do we teach a computer to automatically do it for us? The first step is finding a clear way to represent the text in a numerical fashion. When Luhn first proposed his summarizer, he proposed assigning probabilities to each word within a text. The formula looks like:

$$\text{word probability} = \frac{f(w_i)}{N} \quad (1)$$

where:

$f(w_i)$ = the number of occurrences of a word (w_i) in the text

N = total number of words in the document:

Luhn's motivation behind assigning probabilities was that the significance of a sentence is reliant upon the words it contains [9]. The logic is that important words should appear relatively frequently in a document, and sentences containing important words should also be important. After this data representation was created, Luhn could just select an arbitrary number of sentences ranked by highest average probability to create a summary.

Luhn himself did point out a few caveats to his method. One of these is that words occurring extremely frequently are not necessarily important. These could be stop words such as "the", "as", and "from." They could also be domain-specific words that do not add particular meaning to the text. For instance, in genomic-based research papers, the word "gene" occurs with high frequency. But the word itself does not particularly add meaning to sentences because it occurs so frequently. To resolve the issue of stop words, Luhn proposed a stop word list that would manually remove these kinds of words before probabilities are calculated [21]. To resolve the issue of over-frequent domain-specific words, Luhn selected a cutoff where probabilities below a certain threshold were counted as 0. The idea behind this is that words with low probabilities are probably not informative in terms of subject matter, so removing this "noise" should actually help performance [9].

Overall, Luhn's contribution was extremely imperative because it became the foundation in which all extractive summarizers were built upon.

Of course, Luhn's summarizer is now outdated. As he pointed out himself, there were a few caveats to his method. Fortunately, researchers improved upon Luhn's probability metric to create something called TF-IDF (Term Frequency Inverse Document Frequency) weighting. The notion behind TF-IDF is that important words in a text should occur commonly in one document but seldom occur in other documents. This metric would solve the main caveat of Luhn's original metric by assigning less weight to commonly occurring stop and domain-specific words. In terms of single-document summarizers, TF-IDF is formally defined as [22]:

$$f(w) * \log \frac{|S|}{s(w)} \quad (4)$$

where:

$f(w)$ = frequency of word w in sentence

$s(w)$ = number of sentences in the document containing word w

$|S|$ = the number of sentences in the document

TF-IDF is the most well-known of the metrics. However, there are many other variants for frequency-based metrics. These are topic signatures, TF-IDF variants, etc. More information on how these metrics work can be found here: [22].

2.1.5 SumBasic

SumBasic is a very simple probabilistic method that directly extends upon Luhn's method. It is one of the simplest unsupervised summarizers, and it is commonly used in research papers as a baseline to evaluate state-of-the-art models [18, 19, 20]. Vanderwende et al. proposed the Sumbasic system [11], which uses the word probability approach to produce system summaries .

Step 1: For each sentence S_j , a score is assigned based on the average probability of the words in the sentence.

$$S_j = \frac{\sum_i^N p(w_i)}{N} \quad (3)$$

where:

S_j = SumBasic sentence score

$p(w_i)$ = probability of word (w_i) in the text

N = the number of sentences in the document

Step 2: The sentence with the highest score is the first sentence to be extracted for the system summary.

Step 3: In the next step, this sentence is removed from the text. For each word in the chosen sentence, the probabilities are then updated by squaring the probability of the word:

$$p(w_{i_new}) = p(w_i) * p(w_i) \quad (4)$$

where:

$p(w_{i_new})$ = updated probability of word (w_i)

$p(w_i)$ = current probability of w_i

Step 3 is interchangeable, as users can choose not to update the word probabilities. By opting out of the probability update, SumBasic will end up creating summaries that are more redundant by selecting sentences that are more similar to one another.

Step 4: Steps 1-3 are repeated until an arbitrary number of sentences is selected.

2.1.6 Graph-based Summarizers

Graph-based summarization methods are shown to be very versatile in that they are useful in both single-document and multi-document applications. They generally outperform baseline methods such as SumBasic because of their ability to better incorporate sentence to sentence relationships [21].

The basic idea behind graph-based summarization methods is that sentences more strongly related to other sentences are more important.

To find the most salient sentences, graph-based methods construct a bidirectional graph where sentences form the vertices of the graph while a similarity measure between sentences constitute the edges. This similarity metric is arbitrary, but the standard is to use cosine similarity of TF-IDF word weights. In most online ML packages for graph-based methods, each sentence is represented as a vector with entries populated by TF-IDF values.

For instance, refer to the following sentences below.

“I am doing my dissertation.” $\rightarrow s_1$

“I am doing my thesis.” $\rightarrow s_2$

“Doing my thesis.” $\rightarrow s_3$

These sentences will have a vector representation shown Table 1, where each entry in V_i corresponds to a unique word from the union of all words of s_1, s_2, s_3 :

Table 1: Example Sentence Vectors

	I	am	doing	my	thesis	dissertation
$V(s_1)$	0.176	0.176	0.000	0.000	0.000	0.477
$V(s_2)$	0.176	0.176	0.000	0.000	0.176	0.000
$V(s_3)$	0.000	0.000	0.000	0.000	0.176	0.000

According to [12], graph-based summarizers would then calculate pairwise cosine similarity, which measures the size of the angle between two vectors, defined as:

$$\text{Cosine Similarity}(s_1, s_2) = \frac{V(s_i) \cdot V(s_j)}{\|V(s_i)\| \|V(s_j)\|} \quad (5)$$

where:

$V(s_i)$ = the vector representation of a first sentence (s_i)

$V(s_j)$ = the vector representation of a second sentence (s_j)

If we were to continue with Table 1 and calculate cosine similarity between each pair of sentences, the intermediate matrix representation would look something Table 2:

Table 2: Pairwise Cosine Similarity

	V1	V2	V3
V1	1	0.35	0
V2	.35	1	0.24
V3	0	0.24	1

Of course, all the entries along the diagonal are trivially equal to 1 because a sentence is completely identical to itself. Cosine similarity will yield values between 0 and 1. The closer the values are to 1, the more similar the sentences are. Once this matrix representation of the graph is created, then various summarization techniques can be applied on top of it to create document summaries.

2.1.6.1 LexRank and TextRank

LexRank is one of the most popular graph-based summarizers. Using the matrix representation in Table 2, PageRank is run on the matrix until convergence. In this case, the transition probabilities are the similarity measures and the “nodes” are the sentences. The score assigned to a sentence is obtained by averaging its similarity score (cosine similarity) to every other sentence in the text. The sentences with highest scores are extracted first until an arbitrary sentence limit is reached. More information on the specifics on PageRank can be found in [22]. While there are a few variations of LexRank, one being Lexrank based on degree centrality [7], most open-sourced LexRank packages are based on Continuous LexRank, which is the method we described above.

TextRank is exactly the same as LexRank, but it differs in one way. The difference lies in the similarity metric. While LexRank uses cosine similarity of TF-IDF values, TextRank uses the average number of overlapping words. More details on TextRank can be found in [10].

Overall, LexRank and TextRank are still pretty good algorithms. In Erkan and Rakev’s paper [7], LexRank was shown to significantly outperform the baseline methods of the time. Namely, this was a centroid-based clustering method that was considered to be a very strong baseline. In another paper (*TextRank: Bringing Order Into Texts*), TextRank was also shown to beat various baselines. In this case, all 15 chosen baselines, including the DUC (Document Understanding Conferences) certified baseline of taking the lead-3 sentences of news articles, were significantly outperformed by TextRank [10]. Because both LexRank and TextRank are still relatively competitive and because of their ability to generalize to many kinds of datasets, they are both still used as common baselines. See [18, 19, 20], for their use as baselines for current state-of-the-art techniques.

2.1.7 Topic Based Summarizers

Topic models are algorithms that help identify latent themes in the text. The reason that the themes are “latent” is that the main ideas of the text are not readily apparent at first glance. Topic models can organize the text by this set of “latent” themes, and then indicate to the user what those themes actually are. The value in this approach is that it allows the sentences to be grouped together by their underlying topics, a task that would otherwise require a hefty amount of self-annotation.

There are a few topic-based methods that have gained traction amongst academia. We will introduce two of them here: LSA and LDA. LDA is widely considered the golden standard for extractive, unsupervised summarizers [23]. Algorithmically, LDA is a very complex method with various hyperparameters to tune, and as there is a lack of open-sourced packages that can implement an LDA-based summarizer, manually implementing it is very difficult. Though LDA generally outperforms LSA, LSA is still a very good method. Because there are online resources with documented LDA code, it is also much simpler and easier to implement. Therefore, we will dedicate more time to discuss LSA in this section. If you would like to know more about LDA, information can be found in [4, 5].

2.1.7.1 Latent Semantic Analysis

LSA is a statistical method that determines how related sentences within a text are to one another in order to determine sentence importance. Let’s take an example of a text from our dataset and observe the summary LSA produces.

“Over the past 2 weeks, I have been receiving excessive amounts of telephone calls from the company listed in this complaint. The calls occur between XXXX XXXX and XXXX XXXX to my cell and at my job. The company does not have the right to harass me at work and I want this to stop. It is extremely distracting to be told 5 times a day that I have a call from this collection agency while at work.”

This text can be divided into two parts. The first part generally sets the context of the situation the customer is facing; it provides the reader information about how long the situation has been going on, and how often the situation happens per day. The second part sets the tone for how the customer feels about the numerous phone calls. As you can see, there are two distinct topics. LSA will extract the most important sentence from each topic and provide the summary below:

“Over the past 2 weeks, I have been receiving excessive amounts of telephone calls from the company listed in this complaint. The company does not have the right to harass me at work and I want this to stop.”

The summary does two things very well. It captures the most salient information from the text and simultaneously, it makes the text much more concise. It is worth noting that LSA is not perfect. For instance, the algorithm tends to struggle strongly with texts where most of the sentences are just noise. In these kinds of texts, LSA will mistake unimportant sentences for important ones. This is due to the fact that these unimportant sentences are all strongly related, so they are all assigned higher sentence scores. Nonetheless, LSA is good at removing redundancy, and this trait allows it to create concise summaries that yield better precision.

LSA based summarizers can be broken down into three main steps [2]:

- Input Matrix Creation
- Singular Value Decomposition
- Sentence Selection

2.1.7.1.1 Input Matrix Creation

Here, a matrix with rows representing the words and columns representing the sentences is created. Each column is a one-hot vector representation of each sentence in the text. Table 3 (below) is an input matrix representation of the following sentence.

[‘My name is Pete. Her name is Sarah.’]

	Sentence 1	Sentence 2
Her	0	1
name	1	1
Pete	1	0
My	1	0
is	1	1
Sarah	0	1

Table 3: Example Term-Frequency Matrix

The first column is the vector representation of the sentence “My name is Pete.” The second column is the vector representation of “Her name is Sarah.” The rows are a simple concatenation of all unique words in the text. A value of +1 is assigned to the number of times each unique word from

the text appears in a particular sentence. This is the frequency matrix representation of an input matrix, but there are various types of matrix representations, such as TF-IDF.

MTF (Maximum Term Frequency Normalization): This uses Term Frequency and scales the TF value by dividing by the max-term frequency value (MTF). The MTF is the term in the sentence with the highest Term Frequency. This TF-MTF value is then scaled by additional adjustable constant:

The formula looks like:

$$mtf_i = a + (1 - a) \frac{f(w_i)}{\max(\cup_i^n w_i)} \quad (6)$$

where:

a = arbitrary constant set by the user

$f(w_i)$ = term frequency of word (w_i) in text

$\max(\cup_i^n w_i)$ = maximum term frequency of all words in the text

In TF-IDF, longer sentences generally have higher TF values, simply because they have a higher tendency to repeat the same words again as the sentence length increases. By scaling the TF term by the MTF, we can reduce the weight assigned to longer sentences. This consequently encourages the LSA algorithm to reduce wordiness. For this reason, we will select MTF as our approach when creating the input matrix.

2.1.7.1.2 Singular Value Decomposition

Once the input matrix is created. SVD is performed on the matrix of MTF cell values. SVD is a mathematical method that splits a real matrix into three individual matrices:

$$A = UDV^T$$

- A The original matrix
- U The left singular vectors
- D The diagonal matrix
- V^T The right singular vectors

Here is an example of SVD in the context of our project. Using our example input-matrix from Table 3, this matrix (Table 4) will be decomposed into the U , D and V^T matrices, shown by Table 5, Table 6, and Table 7 respectively:

A-Matrix

	Sentence 1	Sentence 2
Her	0	1
name	1	1
Pete	1	0
My	1	0
is	1	1
Sarah	0	1

Table 4: A-Matrix in SVD

Definition:

The original matrix input matrix.

Dimensions:

- $I \times J$ (I rows, J columns)
- I corresponds to number of unique tokens
- J corresponds to number of sentences
- 6×2 for matrix for Table 4

U-Matrix

	Topic 1	Topic 2
Her	-0.2887	-0.5
name	-0.5773	0
Pete	-0.2887	0.5
My	-0.2887	0.5
is	-0.5774	0
Sarah	-0.2887	-0.5

Table 6: U-Matrix in SVD

Definition:

The U-Matrix contains the relationship between each word topic pair. Large absolute values indicate that the word is strongly related to the topic and vice versa. For instance, “her” has a stronger relationship to topic 2 than topic 1.

Dimensions:

- $I \times J$ (I rows, J columns)
- 6×2 for matrix for Table 6

D-Matrix

	Topic 1	Topic 2
Topic 1	2.449	0
Topic 2	0	1.414

Table 5: D-Matrix in SVD

Definition:

The D-matrix denotes the overall importance of each topic. The values only appear on the diagonals and are sorted in decreasing order. The larger the value, the more important the topic. In the Table 5, topic1 is more important than topic 2, as determined by SVD.

Dimensions:

- $J \times J$ (J rows, J columns)
- 2×2 for matrix above

V^T -Matrix

	Sent 1	Sent 2
Topic 1	-0.707	-0.707
Topic 2	0.707	-0.707

Table 7: V^T -Matrix in SVD

Dimensions:

- $J \times J$ (J rows, J columns)
- 2×2 for matrix above

Definition:

This matrix denotes the relationship between each sentence and each topic. Here, Sent1 and Sent2 are equally related to Topic 1.

2.1.7.1.3 Sentence Selection

After performing both input-matrix creation as well as singular value decomposition, the sentence selection step of LSA takes the V^T Matrix generated by SVD and uses a systematic process to rank sentence importance.

Gong and Liu (2001) [2]

There are various ways to rank sentences. One of the earliest ways to do this is to select the most related sentence to each concept in V^T . This means that we will select the sentence with the highest absolute value for each topic until we reach a predefined number of sentences.

Table 8: V^T -Matrix for Sentence Extraction

	Sent 1	Sent 2	Sent 3	Sent 4
Topic 1	0.54	0.31	-0.22	0.29
Topic 2	0.41	0.56	-0.35	0.44
Topic 3	-0.10	0.11	0.55	0.61
Topic 4	0.01	0.06	0.51	0.43

If we look at a new example of V -Transposed matrix (Table 8) and extract the top 3 sentences, this means we will select “Sent 1” from Topic 1, Sent 2 for Topic 2, and Sent 4 for Topic 3. The shortcoming of this method is that it is impossible to select multiple sentences from the same topic, even if that topic is very important. More often than not, selecting only one sentence from a topic does not give adequate context, leading to incoherent and choppy summaries. While this method ensures that the summary is not redundant by selecting a sentence from each topic, it also encourages the selection of sentences related to unimportant topics. This ultimately can still create summaries with extraneous information.

Steinberger and Jezek (2004) [3]

A better way of sentence selection was proposed by Steinberger and Jezek. This method assigns a score to sentences that gauges how important each sentence is to the entire text. This value is determined by how related each sentence is to each topic.

$$\text{Sentence Score} = \sqrt{\sum_{s=1}^i V_{st}^T * D_{ss}} \quad (7)$$

That is, given an $i \times i$ D-matrix and an $i \times i$ V^T-Matrix, each diagonal entry with indices s, s (row s in D , column s in D) is multiplied by its corresponding indices t, s (column t in V , row s in V). The square root of the summation of each $V_{st}^T D_{ss}$ is taken to obtain a sentence score. An example would clarify the math behind this concept, and we will give an example of sentence scoring in Table 9 and Table 10.

Consider an instance where we have two sentences, so each sentence would be scored in the following way:

Table 9: D-Matrix for Steinberger and Jezek

$D \rightarrow$		Topic 1	Topic 2
	Topic 1	2.449	0
	Topic 2	0	1.414

Table 10: V^T-Matrix for Steinberger and Jezek

$V^T \rightarrow$		Sent 1	Sent 2
	Topic 1	0.707	0.707
	Topic 2	0.707	0.607

Sentence 1: V-Transpose Matrix (1st column in yellow: 0.707, 0.707) is multiplied by the diagonal values of the U Matrix (2.449, 1.414).

Sentence 2: V-Transpose Matrix (2nd column in green: 0.707, 0.607) is multiplied by the diagonal values of the U Matrix (2.449, 1.414).

$$\begin{aligned} \text{Sentence 1 Score} &\rightarrow \sqrt{|0.707 \times 2.449| + |0.707 \times 1.414|} = 1.653 \\ \text{Sentence 2 Score} &\rightarrow \sqrt{|0.707 \times 2.449| + |0.607 \times 1.414|} = 1.609 \end{aligned}$$

Each cell within a sentence vector in the V-Transpose Matrix is multiplied with the cell of the corresponding topic in the U Matrix. The square root of the sum of these values is the score that is assigned to the sentence. As you can see, this method takes into account both the importance of a topic (by incorporating the cell values from the diagonal matrix), and the strength of the relationship between a topic and a sentence (in the V-Transpose matrix). In our example in Table 9 and Table 10, Sentence 1 is determined to be more important and it is accordingly assigned a higher score. Consequently, in the sentence selection process, sentence 1 is extracted first.

This method is advantageous to Gong and Liu's method because it allows multiple sentences to be selected from the same topic. It also gives a better estimate of how related a sentence is to the rest of the

text. Because of these facts, we will use Steinberger and Jezek’s implementation of sentence selection for LSA.

2.1.8 Machine Learning Summarizers

Unlike unsupervised approaches, supervised summarizers differ in a few ways. The first is that they require labels so that the summaries produced can be evaluated against references during training. The second is that in addition to incorporating information among the text data itself, supervised summarizers will self-engineer features to add information that can increase overall predictive power. Some common features are the length of each sentence in the text, the position of the sentence, and the number of capital words in the sentence. These additional details are taken into account by supervised summarizers to produce a prediction of which sentences should make it into the final summary [23]. Common machine learning methods such as decision trees, Naïve Bayes, Hidden Markov Models, Conditional Random Fields, Conditional Random Fields, and Hidden Markov models all produce competitive summaries. Of these, Conditional Random Fields and Markov Models, methods that assume dependencies between other sentences, are the most competitive [23]. The most competitive methods today, however, are the neural based summarization methods used in both extractive and abstractive contexts. Interest in neural nets in general was reignited by the paper, *Attention is All You Need* [16]. This paper describes an attention-based neural method, and researchers decided to use this algorithm on various NLP applications, including summarization, with plenty of success on popular summarization datasets. For instance, in Arumae and Liu’s recent paper [17], they compared the performance of their LSTM-based model with numerous baselines on CNN and Dailymail news data. All neural-related models significantly outperformed the non-neural baselines, excluding the Lead-3 baseline in the CNN news dataset [17]. Other papers where neural models were shown to outperform their non-neural counterparts can be found in [18, 19, 20].

It is worth noting that there is no specific model proposed that gained widespread acceptance and recognition as “the best model” amongst the scientific community. First, the models have only been tested upon very specific data, such as news data. Second, the field is quite new, so these new models have not yet had the time to develop traction amongst academia. To sum it up, there are various kinds of architectures for neural models, and no person in academia can agree upon who’s model is “the best.” To learn more about neural-based summarizers, a few good places to start can be found in [17, 18, 19, 20]. As [17] is the most recent paper of the four, I would start there.

Lastly, one caveat of supervised summarizers is their lack of scalability [23]. Supervised learning tends to require a lot more data to produce competitive results. While obtaining access to plentiful, already-labelled data is not an issue amongst other fields, finding already-labelled entries for text summarization is very laborious. Obtaining annotated entries for summarization would require someone to read thousands of thousands of documents and then manually produce a complete summary. This process is of picking and choosing what content within the summary is important is fairly subjective so there is no guarantee that the annotated entries are completely accurate. Overall, obtaining annotated entries is a fairly time-consuming process, and this becomes an issue all researchers will run into if they want to implement a supervised summarization algorithm.

2.1.9 Automatic Summarization: Closing Statements

I only selected the topics I believed were both more important and relevant for my project. Given the word limit of the project, it is not possible to discuss all possible methods of summarization. If you would like to dive deeper into automatic text summarization, particularly extractive summarization, please refer to [23]. Since more focus was dedicated to unsupervised summarizers, state-of-the-art ML summarizers were not discussed in depth. If you would like to know more about very competitive ML methods, a few good places to start are in [17, 18, 19, 20].

2.2 Evaluation Metrics: Rouge

Rouge is a metric that introduced by the paper in [8], and it quantifies how good a system summary is. To do this, it requires a reference summary, a summary written by a human as a standard of comparison. It then takes the system summary, the summary produced by the machine, and compares how similar the system summary is to the reference. To start off with, there are the classic precision, recall, and f-score rouge measures.

Recall-based rouge measures how much of the reference summary the system summary is capturing. It takes the number of overlapping tokens in both summaries and divides this value by the number of total tokens in the *reference* summary.

$$\text{Recall} = \frac{\# \text{ overlapping words in system and reference}}{\# \text{ of words in the reference summary}} \quad (8)$$

Thus:

- A recall score that is high means that the system summary captures the main idea of the original text pretty well.
- A recall that is low means that the system is missing out on a lot of important details and hence, does not do a great job of summarizing the original text.

Precision-based rouge is the number of overlapping tokens in the system and reference summary, divided by the total number of tokens in the *system* summary.

$$\text{Precision} = \frac{\# \text{ overlapping words in system and reference}}{\# \text{ of words in the system summary}} \quad (9)$$

Because of this denominator, precision penalizes summaries for verbosity. In this way, precision measures both how concise a summary is and how well the system summarizes the text. Thus:

- A precision score that is highest means that the summary is concise and shares a lot of overlapping tokens with the system summary
- A precision score that is medium means that the summary is concise OR has a high token overlap with the system summary
- A precision score that is low means that the summary both is not concise and has low token overlap with the system summary

Both precision-based and recall-based rouge have various uses, but optimal usage is situational. If the user's goal is to ensure that the summary is as holistic and comprehensive as possible, then recall is a

better measure. If the goal is to have both a comprehensive and concise summary, then precision-based rouge may be better in this case. In general, **F1-Rouge** is the most popular metric when quantifying summary proficiency because it strikes a balance between precision and recall. The formula for F1-Rouge is shown below in (10):

$$F1 - Rouge = \frac{2 * precision * recall}{precision + recall} \quad (10)$$

There are many variants of rouge, each quantifying slightly different aspects of the system summary. These rouge variants will be discussed in the following subsection.

2.2.1 Rouge – N

Rouge – N stands for Rouge **N-Grams** where n-grams refers to the number of continuous words. Take a look at some examples:.

Reference Summary:

The cat in the hat.

System Summary:

The cat on the lap.

Rouge-1

Measures the overlap of unigrams, the number of overlapping individual words. In the example above, both the reference summary and system summary have 5 tokens each, 1 token corresponding to each non unique word.

There are 3 overlapping words: “The”, “cat”, and “the”.

There are 5 tokens in the reference summary.

Thus, recall based rouge would yield 3/5.

Rouge-2

Measures the overlap of bigrams, which is the number of overlapping consecutive words. Below are the lists of bigrams:

Reference Summary Bigrams

['The cat', 'cat in', 'in the', 'the hat']

System Summary Bigrams

['The cat', 'cat on', 'on the', 'the lap']

Therefore, there is only 1 identical overlapping bigram, “the cat”.

The number of tokens in the system summary would be 4.

Thus, the rouge-2 metric would yield 1/4

Compared to rouge-1, rouge-2 has finer granularity. Rouge-2 gives a better idea of how similar a system summary is to a reference summary. A higher rouge-2 score means a system summary has sentence-level word order that is similar to that of the reference summary. In other words, rouge-2 is better at indicating how good the flow of a summary is. Therefore, a text that has a higher rouge-2 score could be better than a text with a higher rouge-1 score because it is more coherent and readable. With this kind of logic, rouge-3, the overlap of trigrams, should provide an even better indicator of how

fluent a summary is. The caveat is that as texts get longer, it becomes very unlikely for any two texts, regardless of how similar they are to have many overlapping trigrams. Therefore, N=2 is often the N-gram limit to score system summaries.

2.2.2 Rouge – L (Longest Common Sequence)

Rouge-L solves the Rouge-N issue of infrequent overlapping N-grams for large N. Specifically, Rouge-L measures the longest common sequence of words and removes the consecutive matching requirement in Rouge-N. Instead, it requires in-sequence matching to capture how similar the sentence-level word order between two texts are. Let's take a look at the example below:

S1 S2

System Summary: The cat in the hat. The cat ate the rat

S3 S4

Reference Summary: The cat in the cage. The cat ate rate in the cage.

Rouge-L will take each sentence from the reference and compare it to each sentence in the system, finding the longest common subsequence (LCS) from both sentences and taking the union of them. For instance:

System summary has two sentences: S1 and S2.

Reference summary has two sentences: S3 and S4.

The LCS between S3 and S1 is 'The cat in the' of length 4.

The LCS between S3 and S2 is 'The cat the' of length 2.

The union of these results is 'The cat in the' of length 4.

We then divide 4 by the number of words (5) in reference sentence S3.

Hence, the union LCS of (S3, S1) and (S3, S2) is 4/5.

The LCS between S4 and S1 is 'The cat in the' of length 4.

The LCS between S4 and S2 is 'The cat ate rat' of length 4.

The union of these results is 'The cat ate rat in the' of length 6.

We divide 6 by the number of words (7) in reference sentence S4.

Hence, the union LCS of S4, S1, and S4, S2 is 6/7.

Recall-based LCS would then be:

$$= (4/5 + 6/7) \div (\text{total \# of words in the system})$$

$$= (4/5 + 6/7) \div 12$$

Precision-based LCS would then be:

$$= (4/5 + 6/7) \div (\text{total \# of words in the reference})$$

$$= (4/5 + 6/7) \div 10$$

3 Proposing a New Summarizer

The remainder of this report will discuss the design, implementation, and evaluation of a text summarization method. The first part of section 3 will discuss more details about the data, and how the data was collected. This will be followed by a discussion of baseline selection, where our reasoning behind choosing each baseline is discussed. After this, we will discuss the design and implementation of our summarizer. Finally, we will discuss how good our summarizer is and evaluate its performance on the CFPB dataset.

3.1 Data

As discussed in section 1, this data is from the Consumer Financial Protection Bureau (CFPB). The CFPB provides the data on its website: <https://www.consumerfinance.gov/data-research/consumer-complaints/> [24] and updates daily. The published data only contains the information after the company responds to the complaint, or if a limit of 15 days has been exceeded, whichever comes first.

The Consumer Complaint Database is a collection of complaints about consumer financial products and services that it sends to companies for a response. According to its website, the CFPB analyzes this data to supervise companies, enforce federal consumer financial laws, and write better rules and regulations. To this end, we would like to propose a summarizer that is able to facilitate the execution of these tasks by reducing the amount of time regulators must spend reading overly long customer complaints.

Consumer complaints are generally about the financial troubles the consumer is going through. The CFPB database is exceedingly large, and contains data related to various kinds of products: mortgage, debt collection, credit card, bank account, etc. To reduce the sheer amount of data, we focus solely on the debt collection category and filter out any other entries. Even so, the data is still left with around one million rows.

Within “debt collection” products only, there are a few kinds of issues customers face. Some of these are:

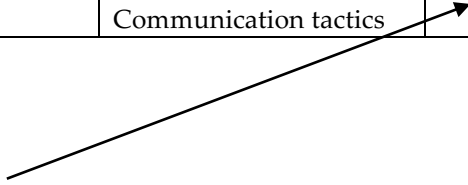
- Attempts to collect debt not owed
- Communication Tactics
- Taking/threatening legal action
- Took or threatened to take negative or legal action
- Disclosure verification of debt
- False statements or representation
- Improper contact or sharing of info
- Continued attempt to collect debt not owed

We randomly sample 4-6 entries from each issue. We also try to select texts of various lengths to increase our chances of getting data that is representative of the dataset. In total, we have 50 entries of data. While a mere 50 entries of data is clearly deficient, allocating more time to collect more data may not be worth it. Namely, in order to properly evaluate the efficacy of our summaries, we need to manually summarize the complaints themselves so that we can properly score them with Rouge. It takes around 24 hours to obtain 50 such summaries.

All in all, a row from the dataset looks like:

Table 11: Example Entry of Data

Product	Issue	Consumer complaint narrative	Complaint ID
Debt collection	Communication tactics		3433198



[Consumer complaint narrative]

“Over the past 2 weeks, I have been receiving excessive amounts of telephone calls from the company listed in this complaint. The calls occur between XXXX XXXX and XXXX XXXX to my cell and at my job. The company does not have the right to harass me at work and I want this to stop. It is extremely distracting to be told 5 times a day that I have a call from this collection agency to stop.”

While this is a short complaint, it can still be summarized, and it is worth noting that most complaints are not this short. By referencing the complaint ID, future researchers can extract the same 50 samples that I used to reproduce my project’s results.

3.2 Baseline Selection

The first step in any research project is to establish baseline values for expected performance. This is important as it provides a threshold of minimum performance our summarizer must meet. This threshold tells us if our summarizer is good or not, and if the summarizer does not exceed this threshold, then we might as well use already-existing, better summarizers.

We select the following methods as our baselines:

- **SumBasic:** See section 2.
- **TextRank:** See section 2.
- **KLSum:** This is an algorithm that selects sentences by minimizing the Kullback-Leibler (KL) divergence between an assumed probability distribution over the words of the text. More details on KLSum can be found in [23].
- **LSA:** See section 2.
- **Random:** This method randomly selects half of the sentences from the text. The purpose of this summarizer is to provide the absolute minimum performance any summarizer must have on the dataset. A summarizer with performance below this threshold is simply not worth consideration.

Since we only have 50 samples of data for our project, pursuing a machine learning model is impractical. We will therefore focus on developing an unsupervised model. For the same reason, we will also compare our model to widely-accepted unsupervised methods. We choose SumBasic, TextRank, KLSum, and LSA as our baselines. We select these methods because:

- There are easily accessible packages able to implement these algorithms.
- Even though these methods are not considered cutting-edge, they still have competitive performance on the most popular summarization datasets, and are still used as common baselines in research papers discussing the creation of new neural models. See [18, 19, 20] for examples of their usage.
- Because they are unsupervised, they generalize better to all kinds of text.
- These models are not as strongly reliant on the sample size of data as supervised summarizers are. They can still be relatively competitive even with severe data constraints.

We use Sumy [13], a MIT-licensed package for python, to implement these summarizers. Sumy will break the summarization process into three steps:

- **Preprocess the Text:** Tokenization of the text will occur, punctuation is removed, and all words are lowercased. Stop words, stemming, or lemmatization are all available options and can be implemented under the user's discretion.
- **Fitting the Model:** Sumy will then use the indicated model to internally rank the sentences and then store the ranks.
- **Sentence Extraction:** Lastly, Sumy will return the top n-sentences after the user specifies n.

For each baseline method, we experiment with various usage of stemming and stop word removal. We then choose the model that yields that highest average performance on all 50 entries of the data. Finally, we compare the best-performing models for each method. In order to ensure equal comparison, each model extracts only half of the sentences of a given text. This number can be changed but as a rule of thumb, 50% tends to yield optimal performance for most summarizers on this data. There is also little point in finding the optimal summary length for each summarizer because our sample size is so small. This means that there is little guarantee that models with summary lengths optimized for our 50 CFPB samples will also yield good performance on the entire dataset. The rouge-scores obtained are shown in Table 12:

Table 12: Baseline Rouge-Scores

Rouge F Scores			
	Rouge 1	Rouge 2	Rouge L
Random	26.9%	6.6%	15.7%
SumBasic	27.7%	6.8%	17.0%
TextRank	26.5%	7.1%	16.3%
KLSum	25.7%	6.0%	16.2%
LexRank	27.5%	7.5%	17.7%
LSA	28.8%	7.9%	18.3%

As you can see above, all the unsupervised summarizers outperform *Random* in every department. Latent Semantic Analysis has the best performance of all summarizers, beating other summarizers in

Rouge-1, Rouge-2, and Rouge-L. KLSum, on the other hand, has appallingly bad performance. It barely outperforms *Random* in Rouge-L, and is worse than *Random* in both Rouge-1 and Rouge-2.

3.3 Creating a New Summarizer

In this section we will discuss the specifics of our summarizer. In particular, we will talk about:

- The motivation behind our summarizer
- The mathematics behind the model

Instead of creating a completely new summarizer from scratch, it is more efficient to build upon already established algorithms. With this in mind, our summarizer will combine elements of LSA and elements of SumBasic. We select LSA because it is by far our best performing summarizer. Logically speaking, building on top of something that already yields good results should in turn yield even better results. We choose SumBasic as our second summarizer. Although LexRank has better performance, the differential is not substantial. Moreover, SumBasic is a simpler algorithm, so it is much easier to work with.

We will build upon code written for the Sumy package [13], available on <https://github.com/miso-belica/sumy>, to implement our own algorithm.

3.2.1 Modifying LSA

Beginning in this section, we will propose a summarizer that will systematically remove the least important sentences from a text first. Then, a summarization algorithm can be applied on top of this trimmed text to produce a summary. The idea behind this method is that by trimming the text of “outlier” sentences, we are removing noise from the data which can confuse traditional summarizers. This can ultimately help summarizers make better decisions, allowing them to produce better summaries.

Therefore, the overall structure of our summarizer will be described by Figure 1:

Figure 1: Model Pipeline



Now that we have explained the motivation behind our summarizer, the first step is to determine a systematic way to actually trim the sentences. To this end, we propose *LSAPlus*, a method based on LSA. In general, LSAPlus will follow all the usual steps of LSA. After the SVD step, things become different:

- 1 At this point, *LSAPlus* will look at the first row of the V^T . Recall that the first row contains the numerical relationship between the most important topic and each sentence.
- 2 It will then rank each sentence, assigning higher priority to sentences more strongly related to the most important topic.

Table 13: V^T -Matrix for LSA-Plus

	Sent 1	Sent 2	Sent 3	Sent 4
Topic 1	0.54	0.31	0.22	0.29
Topic 2	0.41	0.56	0.35	0.44
Topic 3	0.10	0.11	0.55	0.61
Topic 4	0.01	0.06	0.51	0.43

For instance, if the above figure is our V-transpose matrix, then LSA-Plus will *only* look at topic 1 and rank the sentences. In this case, they will be ranked in the following order: Sent 1, Sent 2, Sent 3, Sent 4. In other words, the least important sentences, arranged in order from least

The idea is that by finding the sentences most strongly related to the most important topic, we can simultaneously figure which sentences are the least important. Once we ascertain which one of the sentences are indeed the least important, we can systematically remove them from the text.

3.2.2 Modifying Sumy’s SumBasic (SumPlus)

LSAPlus is not a foolproof method. In fact, *solely* using LSAPlus as our method of trimming texts will often end up removing important sentences. To avoid this kind of undesirable situation, we will need a second opinion to determine whether a sentence should actually be removed from a document. To this end, we propose using SumBasic as our second opinion.

The current implementation of SumBasic by Sumy is with the probability update (see section 2). We change this so that it will not update word probabilities. By disallowing this update, SumBasic will be more inclined to select sentences with greater similarity overlap. The assumption is that sentences more related to other sentences are more significant. Thus, in the same fashion as LSAPlus, we can hone in on unimportant sentences by figuring out which sentence are actually important.

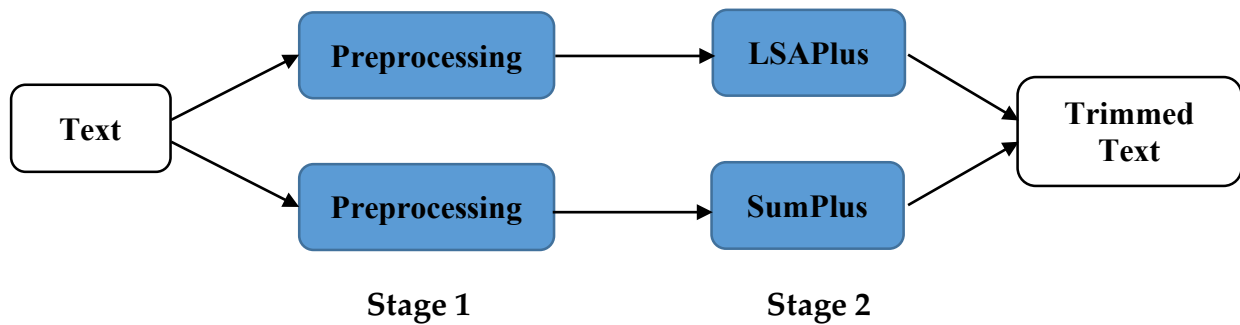
3.2.3 Combining LSAPlus and SumPlus

Now that we have two methods of sentence removal, in this section we will define a system that will incorporate both LSAPlus and Sumplus to make a joint decision. Below we will describe this system in detail.

Beginning with the complaint text, both LSAPlus and SumPlus will produce separate sentence rankings, where the sentences are sorted in decreasing order of importance. We will take the bottom half of both sets of rankings. *Only* sentences appearing in the bottom half of *both* will be eliminated. While taking half of the sentences may seem a bit like overkill, taking half is actually a very conservative measure. Because LSAPlus and SumPlus have significantly dissimilar sentence-ranking approaches, the final rankings produced are ultimately very different. This means that the number of sentences appearing in both the bottom half of rankings for LSAPlus and SumPlus tend to be very small. The output of this entire process is a filtered version of the original text, with a few sentences removed.

Figure 2 below recaps the steps of the model we have described so far:

Figure 2: Model Recap



In stage 1, the text is preprocessed (removing stop words, punctuation, etc). In stage 2, LSAPlus and SumPlus will produce separate sentence rankings. Then, they will consult one another to jointly decide which sentences to remove. The output of stage 2 is a shortened version of the original text, which we call “Trimmed Text” in Figure 2.

It is worth noting that the system we have described so far is merely sentence removal method. It is meant to remove noisy data so that actual summarizers can function more effectively. It is not intended to act as a replacement for these summarizers in any way or form.

To clarify how the system we described so far works, let’s take a look an example text:

- (1) Peter rabbit was a fast runner. (2) Turty the turtle is a slow runner. (3) But they decided to race one another despite one being slower and one being faster. (4) Turty the turtle won the race even though he was slower. (5) **The audience watched the entire thing.** (6) *I ate tacos while I did my homework but I like slow turtles, my pet turtle is named turty.*]

There are two odd sentences. Sentence 6 is clearly a different topic and simply does not belong to the text. Sentence 5 is also unrelated but to a lesser extent.

As an example, let’s say **LSAPlus** would rank the sentences in the following order:

Ranks:

1. (3) But they decided to race one another despite one being slower and one being faster.
2. (4) Turty the turtle won the race even though he was slower.
3. (2) Turty the turtle is a slower runner.
4. (1) Peter the rabbit is a fast runner.
5. (6) I ate tacos while I did my homework but I like slow turtles, my pet turtle is named turty.
6. (5) The audience watched the entire thing.

Let’s say **SumPlus** would rank the sentences in a different way:

Ranks:

1. (3) But they decided to race one another despite one being slower and one being faster.
2. (4) Turty the turtle won the race even though he was slower.
3. (6) I ate tacos and I am a slow worker but I like slow turtles, my pet turtle is named turty.
4. (2) Turty the turtle is a slower runner.

5. (1) Peter the rabbit is a fast runner.
6. (5) The audience watched the entire thing.

Thus, by taking the bottom half of both LSAPlus and SumPlus, respectively, we have:

LSAPlus Bottom Ranks

- (1) Peter the rabbit is a fast runner.
- (6) I ate tacos while I did my homework but I like slow turtles, my pet turtle is named turty.
- (5) The audience watched the entire thing.

SumPlus Bottom Ranks

- (2) Turty the turtle is a slower runner.
- (5) The audience watched the entire thing.
- (1) Peter the rabbit is a fast runner.

If we look at the rankings above, because only (1) and (5) appear in *both* sets of rankings, only these sentences are removed. (2) and (6) will be kept, as there is no unanimous consensus that these sentences are not important. The modified output will look like:

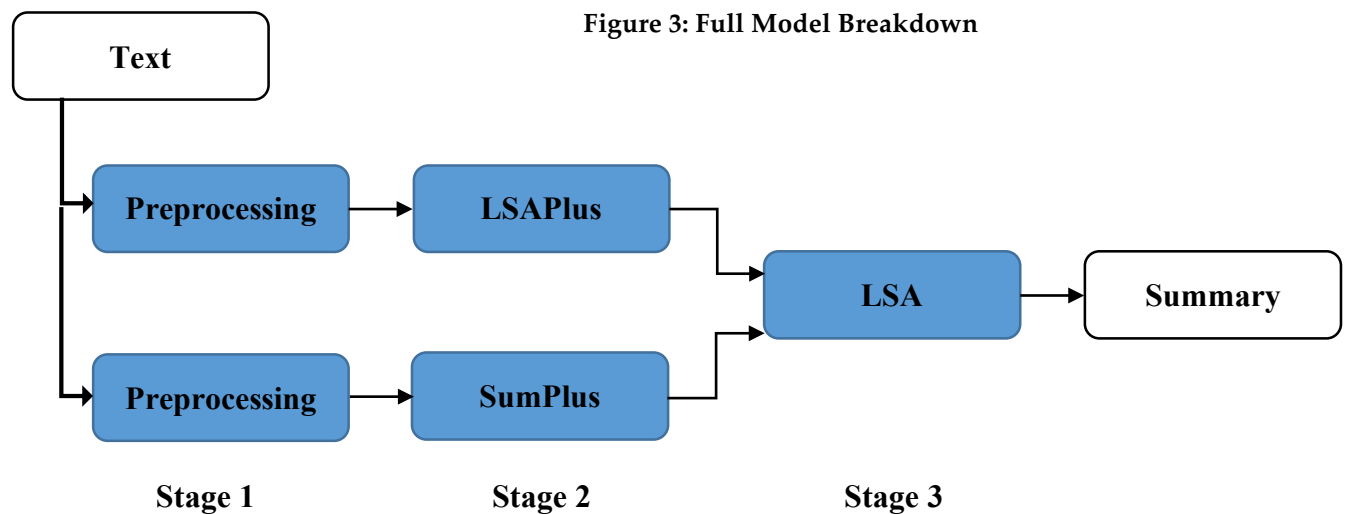
'Peter rabbit was a fast runner. Turty the turtle is a slow runner. But they decided to race one another despite one being slower and one being faster. Turty the turtle won the race even though he was slower.'

This output will then be fed to an algorithm where the actual summaries will then be produced. This is described in the next section.

3.2.4 Reapplying LSA

Generally, because LSAPlus + SumPlus only reduce the original text by a few sentences, the output is by no means a summary. To produce an actual summary, we choose LA as our summarizer. The reason for our choice is that LSA has by far the best baseline performance on our dataset.

Putting it all together, our model will be broken down into the following steps:



Stage 1: The text for LSAPlus and SumPlus are separately preprocessed. This is because Sumbasic seems to function better with stop word removal while LSA seems to do better without stop word removal.

Stage 2: LSAPlus and SumPlus rank the sentences separately. Then they jointly deliberate to determine optimal sentence removal.

Stage 3: LSA summarizes the output from stage 2. It yields final summaries that are 50% the length of the modified text. The reasoning behind 50% was explained in the beginning of Section 3.2.

We will call the entirety of this process, that is, (Stage 1 + Stage 2 + Stage 3), **LSA-Pro**.

4 Evaluation

In this section we evaluate how good our summarizer. Particularly, we discuss the following topics, in no particular order:

- How functional is our summarizer: does it outperform all of the baselines we have selected?
- In what ways is it better than current baselines, and in what ways is it worse?
- What adjustments can be done to improve our summarizer?
- What are the limitations of our evaluation method?

To properly evaluate our summarizers, the system summaries our evaluated against reference summaries in which I hand-labelled. That is, I went through every single one of the entries in the dataset and wrote summaries I thought best reflected the sentiment of the original text. There are a few concerns I would like to voice before presenting the results of my evaluation. First, the performance is dependent on the summaries I produce. By no means do I write optimal summaries, since summaries are often objective. In most research settings, multiple volunteers all write their own summaries, so a system summary is scored by averaging all of its scores based on the different references. However, I worked on this project individually, and did not have volunteers to help me. Second, because the sample size of our data is quite small, the results are likely subject to some variation. With these concerns in mind, let us proceed and observe the results given by the Rouge-1, Rouge-2, and Rouge-L metrics.

Table 14: Comparing Rouge-1 for LSA-Pro Against Baselines

Rouge F Scores			
	Rouge 1	Rouge 2	Rouge L
Random	26.9%	6.6%	15.7%
SumBasic	27.7%	6.8%	17.0%
TextRank	26.5%	7.0%	16.3%
KLSum	25.7%	6.0%	16.2%
LexRank	27.5%	7.5%	17.7%
LSA	28.8%	7.9%	18.3%
LSA-Pro	29.4%	7.9%	18.5%

As shown in Table 14, our summarizer, LSA-Pro is better in every category. The key thing this table illustrates is the disparity between LSA-based summarizers and everything else. LSA-based summarizers lead by at least 1 percentage points in Rouge 1 compared to all other summarizers.

Figure 15 is a table containing the performance of summarizers by sentence length:

Table 15: Rouge-1 Scores for Various Summarizers by Text Length

Rouge-1 F Scores				
Summary Length	Short (1-10 Sent)	Medium (10-15)	Long (15-20)	Very Long (>20)
SumBasic	27.5%	26.0%	31.4%	24.5%
TextRank	27.3%	24.4%	27.3%	28.4%
KLSum	24.5%	25.1%	28.7%	24.4%
LexRank	27.8%	26.6%	27.9%	28.5%
LSA	29.0%	25.9%	32.5%	28.4%
LSA-Pro	29.5%	28.6%	31.3%	26.5%

The highlighted entries display the summarizers with the best performance by sentence length for various texts. As you can see in Figure 15, LSA-Pro does not have the best performance across all categories. It is second best in summarizing long texts and it is 3rd worst in summarizing extremely long texts.

However, LSA-Pro has the best performance in summarizing short and medium texts. Here, it has a 0.5% differential between itself and the next best summarizer. In addition, it crushes every other summarizer by at least 2% for medium length texts.

Given these results, it is worth examining why LSA-Pro has better performance on short and medium length texts and worse performance on long and very long texts, as compared to standard LSA.

Table 16: Comparing Rouge-1 for Short Summaries

Rouge-1 Scores: Short Summaries			
Summary Length	F-Score	Precision	Recall
LSA	29.0%	23.2%	47.1%
LSA-Pro	29.5%	28.3%	39.1%

As you can see in Figure 16, LSA-Pro has better precision: this makes sense as the algorithm first gets rid of the least desirable sentences and then selects the top half of the remaining sentences. In comparison, standard LSA simply selects the top half of the remaining sentences without removing any in the first place. Thus, LSA-Pro removes more sentences and creates summaries that are shorter than those created by standard LSA. So at the cost of having better precision, LSA-Pro has worse recall. But, the difference in precision is more than enough to compensate so that the overall F1 Score for LSA-Pro is better.

Table 17: Comparing Rouge-1 for Medium-Length Summaries

Rouge-1 Scores: Medium Length Summaries (5- 10 Sentences)			
Summary Length	F-Score	Precision	Recall
LSA	25.9%	18.8%	46.4%
LSA-Pro	28.6%	21.8%	45.8%

The same story is true for medium length summaries in Figure 17: our summarizer has much better precision, and this precision more than compensates for the slight drop in recall.

Table 18: Comparing Rouge-1 for Long Summaries

Rouge-1 Scores: Long Summaries (10-15 Sentences)			
Summary Length	F-Score	Precision	Recall
LSA	32.5%	26.5%	46.4%
LSA-Pro	31.3%	25.4%	45.2%

Figure 18 shows that while *Stage 2* of LSA-Pro seems to be effective on short and medium length texts, it seems to struggle on long texts. This is counterproductive because the whole purpose of implementing *Stage 2* of LSA-Pro was to remove the noise from the data to better enable LSA to carry out its functions. What instead seems to be happening is that there is in fact too much noise, so *Stage 2* (LSAPlus + SumPlus) has hard time of determining which sentences are truly worth removing.

The same occurs to a more extreme extent for very long sentences, shown by the Figure 19:

Table 19: Comparing Rouge-1 for Super Long Summaries

Rouge-1 Scores: Super Long Summaries (>20 Sentences)			
Summary Length	F-Score	Precision	Recall
LSA	28.4%	22.9%	41.3%
LSA-Pro	26.5%	21.5%	38.9%

Both the precision and recall are worse. While this data is small (4 entries), it is still a cause for concern. It is more evidence showing that *Stage 2* of LSA-Pro struggles with longer texts because of the additional noise present, making it more difficult to distinguish between what is actually significant and what is not.

A simple solution would be to implement LSA-Pro on shorter texts (≤ 15 sentences) and standard LSA on longer texts (> 15 sentences). Indeed, changing our implementation will improve the results on the dataset we are given, but it is worth noting that slight data snooping is occurring. In reality, we do not have access to the entire dataset. Although it is indicated making this change will have an improvement on the real data, it is not guaranteed. Nevertheless, because our dataset is so small, we have no choice but to take a leap of faith. Therefore, we will modify our algorithm so that LSA-Pro is used on shorter texts and standard LSA is used on longer texts. Table 20 shows the results of this modified summarizer:

Table 20: Comparing Rouge Scores for Final Summarizer (LSA-Pro + LSA)

Rouge F Scores			
	Rouge 1	Rouge 2	Rouge L
Random	26.9%	6.6%	15.7%
SumBasic	27.7%	6.8%	17.0%
TextRank	26.5%	7.0%	16.3%
KLSum	25.7%	6.0%	16.2%
LexRank	27.5%	7.5%	17.7%
LSA	28.8%	7.9%	18.3%
LSA - Pro	29.4%	7.9%	18.5%
LSA-Pro + LSA	29.9%	8.3%	19.0%

According to Table 20, LSA-Pro + LSA has the best performance across all Rouge types. It is now 1.1 percentage points better than LSA in Rouge-1. Additionally, it now has better performance than LSA in both Rouge-2 and Rouge-L. Furthermore, it now also exhibits the best performance across texts of all lengths for Rouge-1 (Table 21 below).

Table 21: Comparing Rouge-1 Scores for Final Summarizer by Text Length

Rouge-1 F Scores				
	Short	Medium	Long	Very Long
SumBasic	27.5%	26.0%	31.4%	24.5%
TextRank	27.3%	24.4%	27.3%	28.4%
KLSum	24.5%	25.1%	28.7%	24.4%
LexRank	27.8%	26.6%	27.9%	28.5%
LSA	29.0%	25.9%	32.5%	28.4%
LSA-Pro	29.5%	28.6%	31.3%	26.5%
LSA-Pro + LSA	29.5%	28.6%	32.5%	28.4%

There are quite a few drawbacks in our analysis of the model. These drawbacks are all related to the limited sample size of our dataset in the first place. To begin with, the cutoffs for short, medium, long, and super long texts are quite arbitrary. Depending on where one chooses to set these cutoffs, the results could in fact be different. By breaking the text into 4 distinct categories in the analysis phase, it further decreases the size of these samples. Therefore, it is possible that the conclusions drawn here may not necessarily hold true on the entire dataset.

A way to circumvent this issue to better evaluate the performance of our model is by randomly dividing the data into 4 sets of text. By inducing randomness, we are simulating how our summarizer will perform on the actual data itself.

Table 22: Comparing Rouge-1 Scores for Final Summarizer by Various Data Splits

Rouge-1 F Scores					
	Split 1	Split 2	Split 3	Split 4	Average
SumBasic	31.2%	26.3%	25.3%	18.4%	25.3%
TextRank	31.1%	25.5%	25.3%	24.4%	26.6%
KLSum	29.8%	26.4%	22.3%	24.7%	25.8%
LexRank	32.1%	26.4%	25.6%	26.3%	27.6%
LSA	31.3%	28.9%	26.8%	28.6%	28.9%
LSA-Pro	31.6%	27.3%	30.0%	28.8%	29.3%
LSA-Pro + LSA	31.9%	27.3%	30.0%	30.2%	29.9%

The data displayed by Figure 22 displays promising results. In 2 of the 4 splits of data, our final model, LSA-Pro + LSA displays the best performance. In Split 2, although (LSA-Pro + LSA) does not display the best performance, it still has the 2nd best performance among all summarizers. In Split1, it is damn near best, only lagging 0.2% behind TextRank. Overall, LSA-Pro + LSA has the best average performance, and it is significantly better than SumBasic, TextRank, KLSum, and LexRank by a long shot. These results are reassuring; it provides some confidence that our summarizer will generalize well.

While we have established that our summarizer is indeed better than the widely-accepted baselines, another important part of summary evaluation is functionality. Just because our summaries have better Rouge performance is not a complete promise that the summary itself is good. Therefore, the question we try to answer in this section is: based on our own assessment, how functional are the summaries we produce?

To this end, we will step out of the bounds of Rouge and personally evaluate an example of a decent system summary and a bad system summary to give an idea of how good our summarizer actually is.

Reference (Original Text)

"I had my vehicle repoed & I had to use XXXX paychecks to get it back & now I've gotten behind on my loans. I'm job hunting also for a second income to get caught up since my husband passed. I have references on a loan with Loan Express & a named XXXX that works there is constantly calling all of my references & my work supervisor on a daily basis even when they 're at work or home repeatedly & also leaving message on their phones on a daily basis. I sent them a letter stating my references names & phone numbers & asked them to please stop calling these people constantly. However my in law told me that yesterday he called her while she was at work repeated so much until she stopped answering because he started using other phone numbers to get her to answer & my other references say the same has happened to them too. Also, I don't know why he's calling my supervisor but it has gotten me in trouble at work & he wants him to stop leaving him message also."

System Summary

"I'm job hunting also for a second income to get caught up since my husband passed. I have references on a loan with Loan Express & a man named XXXX that works there is constantly calling all of my references & my work supervisor on a daily basis even when they're at work or home repeatedly & also leaving messages on their phones on a daily basis. Also, I don't know why he's calling my supervisor but it has gotten me in trouble at work & he wants him to stop leaving him messages also."

In my opinion, the system summary is decent. The summarizer removes extraneous information such as, “I had my vehicle repoed...”. It also removes redundant information. For instance, the customer provides multiple examples of how her contacts are being harassed by the debt collector, citing both calls to her in-law and her supervisor. The summary does not need to include all of these examples to be informative. In light of this, the algorithm selects just one of these examples to include in the summary. By removing extraneous and redundant information, the algorithm produces a summary that is shorter but just as revealing as the original text.

Below, we present an example of a bad summary produced by our summarizer.

Reference

“I received phone calls from a company called Monarch Recovery quite a number of times. The company has never made any mention of any amount specifically, however, I continued to receive more phone calls from this company. It became so bad that the company not only used the phone number listed initially, but retained several other numbers in order to continue to harass me, threatening me with legal actions. The additional phone numbers that were associated with the company are: XXXX and XXXX.”

System Summary

“I received phone calls from a company called Monarch Recovery quite a number of times. The company has never made any mention of any amount specifically, however, I continued to receive more calls from this company.”

The summary makes it sound like the collection company only called the customer a handful of times. By choosing not to include essential details such as “...threatening me with legal actions,” the summary fails to illustrate the overall magnitude of the situation. A better summary would look like:

“I received phone calls from a company called Monarch Recovery quite a number of times. It became so bad that the company not only used the phone number used initially, but retained several other numbers in order to continue to harass me, threatening me with legal actions.”

In conclusion, our summarizer is generally decent. It struggles in a few areas:

- 1) Texts that are extremely short are already concise in nature. It may not be necessary to summarize them. By doing so, we risk losing a lot of important information.
- 2) Summaries for longer texts may suffer a bit from readability issues. One sentence may not lead to the next so the flow of the summary may feel choppy. However, the salient information in longer summaries is generally included.
- 3) Our summarizer is extractive. This becomes an issue when a sentence includes both important and irrelevant information. It cannot pick and choose what part of the sentence to extract. If there is not a better sentence ranked before it, our summarizer is forced to select the entire sentence.

Overall, my personal assessment of my summarizer (LSA-Pro + LSA) is that it is functional. It produces summaries that are both more concise and still understandable. It does not, however, produce *optimally* readable summaries. This is an area for future exploration: with a lot more data and more powerful computational tools, a more advanced summarizer can be built to fulfill this goal.

5 Conclusions and Future Steps

I have determined a few important areas for further exploration. First, future research that can duplicate this study's approach, while significantly expanding the size of the dataset by hand-labelling summaries, would yield more conclusive evidence for the effectiveness of the LSA-Pro + LSA algorithms compared to the standard baselines.

Furthermore, I created summaries equal to 50% of the total number of document sentences. A good area for future study is to find what effect summary length will have on overall performance for different algorithms. This study would be incredibly beneficial to the general public at large. Finding the optimal summary/document length ratio would be a game-changer because there is currently not a lot of research done in this area.

In this project, I proposed a unique way to automatically generate summaries. This method relies upon:

- Developing a systematic way to remove the text of important sentences (*Stage 2* of LSA-Pro)
- Feeding the *Stage 2* output into a standard summarization algorithm (LSA)

Section 4 shows that *Stage 2* of LSA-Pro has mixed results. For short to medium length texts, *Stage 2* seems to lead to considerable improvement in performance. However, for long and very long texts, *Stage 2* actually seems to cause a drop in performance. Of course, these results are unique to the CFPB dataset. I would therefore be cautious when generalizing these findings to other kinds of data.

Even though *Stage 2* of LSA-Pro did not have good performance on long texts, I still think that there is a lot of potential for approaches like it. This approach is unique in the sense that most summarizers take the entire text as input, whereas LSA-Pro takes an already-filtered version of text. Summarization algorithms often have to deal with tons of text data, and finding a way to pre-trim the data will reduce the total amount of data summarizers have to sift through. This can not only speed up training time, but can also improve predictive power by removing noise. Because of these facts, I believe there are a lot of opportunities in this area and would encourage future researchers to explore this field further.

6 Self-Assessment

6.1 Strengths

Magnitude of Research: I spent a lot of time going through various research papers, all of which gave me a very strong foundation of the automatic text summarization field in general. From these papers I not only learned about important summarization methods, but I also learned how to present my project in a way that shows how impactful my findings are. Lastly, because my research for this project was very holistic, I truly believe I was able to provide a very comprehensive overview of the field of automatic text summarization, and this helped set the stage for me in the rest of this project.

6.2 Weaknesses

Organization: My code was all over the place. All too often it felt that I spent a bit too much time looking for specific parts of my code that I needed. Had I been more organized, I could have found relevant material more efficiently.

Insufficient Background: Although I am quite familiar with neural networks, my understanding of the most cutting edge neural nets in the context of field of natural language processing is lacking. While doing my research for this project, I encountered many research papers all dealing with neural based methods for document summarization. These papers often discussed encoder decoder frameworks or transformer/attention-based methods for document summarization. Although implementing neural nets for my project was impractical, my limited understanding of these cutting edge methods also played a role in dissuading me from pursuing a neural based method in the first place.

Lack of Data: Because the sample size of my data is so small, I think it limited the scope of my project. One way in which it limited me was that it forced me to use an unsupervised method of summarization. It is not practical to pursue a supervised way of summarizing with a mere 50 labels. This ultimately discouraged me of using a machine learning method, which has more applications in industry. Second, having such a lack of data makes it extremely difficult to draw reliable conclusions. Third, I spent a good amount of time trying to labeling data. While I did learn about the data during this process, time may be better spent elsewhere. Lastly, my dataset is not well-known in either the academic and professional setting. If I were to have a job interview or discussion with academia, telling these people that I created a summarizer that was able to perform well on a random dataset would not seem like much of a feat, since the dataset itself is relatively unpopular in the summarization field at large. Next time, I will be sure to spend a lot of time researching good datasets so I can have more flexibility in my project.

7 Professional Issues: Transparency and Confirmation Bias

7.1 Transparency

Transparency refers to being honest with all the results of your project, good or bad. Non-transparency can manifest in many forms. Perhaps one of the most common ways it can manifest is in the cherry-picking of data. Cherry-picking is the willful selection of only the data that supports your hypothesis, and a blatant disregard for the data that does not support the same hypothesis. Cherry-picking data and presenting these results to clients, for instance, is extremely unprofessional because it can cause the clients to make incorrect decisions they would otherwise not make.

In my project, there were multiple times where I could have cherry-picked my data to manipulate the results in favor of the hypothesis. Table 23 shows what these results would have looked like if I had done so:

Table 23: Cherry-picking Rouge Scores

Rouge F Scores	
	Rouge 1
SumBasic	28.9%
TextRank	28.1%

KLSum	27.5%
LexRank	29.4%
LSA	29.8%
LSA-Pro	31.5%

This is better than the original result. Instead of a mere 1% difference, LSA-Pro is now an entire 1.7% better than any other summarizer in Rouge-1. In this hypothetical situation, I committed two offences. The first is that the Table 23 results are calculated on only half of the data, particularly the data that inflates the Rouge-1 score of LSA-Pro. The second is that results for Rouge-2 and Rouge-L are intentionally left out. They are left out because they are essentially equal for LSA and LSA-Pro, and this would be evidence refuting the effectiveness of my method. The combination of these two offences clearly adds a sense of unwarranted confidence by exaggerating the performance of my summarizer.

If I were to be completely transparent with my findings, however, I would show the following graph, which was shown in Section 4:

Table 24: Rouge-Scores of Entire Dataset

Rouge F Scores			
	Rouge 1	Rouge 2	Rouge L
Random	26.9%	6.6%	15.7%
SumBasic	27.7%	6.8%	17.0%
TextRank	26.5%	7.0%	16.3%
KLSum	25.7%	6.0%	16.2%
LexRank	27.5%	7.5%	17.7%
LSA	28.8%	7.9%	18.3%
LSA-Pro	29.4%	7.9%	18.5%

Clearly, the differential in Rouge 1 is now a lot smaller. Moreover, the Rouge 2 and Rouge L scores are displayed. This provides a much more truthful and holistic picture of how my summarizer fares against the widely-accepted baselines.

7.2 Confirmation Bias

A common theme among all researchers in every field is confirmation bias, defined as the propensity to interpret new evidence as validation for one's own beliefs. Like cherry-picking, it causes the researcher to deceive his or her clients into making bad decisions. Unlike cherry-picking, however, confirmation bias is different in that it is more of unintentional rather than an intentional state of mind. This state of mind is arguably worse than cherry-picking because it is something that people are all too often unaware of, and you cannot fix something if you do not know it is an issue in the first place.

I personally think that confirmation bias is dangerous because it can prevent a researcher to overlook the weaknesses his or her model. This ultimately prevents the researcher from making improvements, which is the whole premise of doing research. Though I am critical of people who have confirmation bias, I too, almost fell victim to it. Below I explain how I suffered from confirmation bias in the context of

Tables 25-28, and how I was fortunately able to catch myself in the act before making any further mistakes.

Table 25: Rouge-1 Scores for Long Summaries

Rouge-1 Scores: Long Summaries (10-15 Sentences)			
Summary Length	F-Score	Precision	Recall
LSA	32.5%	26.5%	46.4%
LSA-Pro	31.3%	25.4%	45.2%

Table 26: Rouge-1 Scores for Long Summaries

Rouge-1 Scores: Very Long Summaries (>20 Sentences)			
Summary Length	F-Score	Precision	Recall
LSA	28.4%	22.9%	46.4%
LSA-Pro	26.5%	21.5%	38.9%

At first glance, it is quite clear that standard LSA outperforms our summarizer, LSA-Pro, for both long summaries and very long summaries. However, I was first inclined to disregard this information because it is not favorable to the hypothesis that LSA-Pro is a better summarizer.

Table 27: Rouge-1 Scores for Short Summaries

Rouge-1 Scores: Short Summaries			
Summary Length	F-Score	Precision	Recall
LSA	29.0%	23.2%	47.1%
LSA-Pro	29.5%	28.3%	39.1%

Table 28: Rouge-1 Scores for Medium-length Summaries

Rouge-1 Scores: Medium Length Summaries (5- 10 Sentences)			
Summary Length	F-Score	Precision	Recall
LSA	25.9%	18.8%	46.4%
LSA-Pro	28.6%	21.8%	45.8%

This is a classic example of confirmation bias. I was willing to accept the favorable Rouge-1 scores for short and medium summaries but at the same time, I was not willing to accept the Rouge-1 scores for long and very long summaries. In my mind, I rationalized the inferior rouge-1 scores for LSA-Pro as a result of an unlucky split of data. Fortunately, I thought about it and reasoned that under this logic, one could also make an argument that the favorable Rouge-1 scores for short and medium summaries was a fluke. I realized that I was going out of my way to find results that would support my hypothesis and was ignoring the results unfavorable to it.

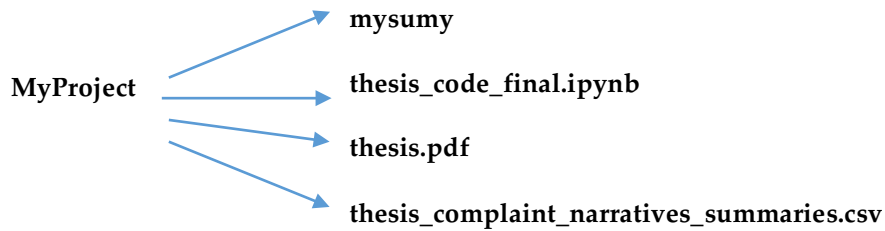
Because I caught myself in the act, I was able to fully acknowledge where my model was weak. In doing so, it allowed me to make the necessary adjustments to improve its performance. This particular adjustment was using LSA-Pro for short and medium texts, and using LSA for long and super long texts. Had I suffered from confirmation bias, I would not have had been able to make these changes.

References

- [1] Makbule Gulcin Ozsoy, Ilyas Cicekli, and Ferda Nur Alpaslan. 2010. Text summarization of turkish texts using latent semantic analysis. In Proceedings of the 23rd international conference on computational linguistics. Association for Computational Linguistics, 869–876.
- [2] Y. Gong and X. Liu “Generic text summarization using relevance measure and latent semantic analysis,” in Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, ser. SIGIR ’01. New York, NY, USA: ACM, 2001, pp. 19–25. [Online]. Available: <http://doi.acm.org/10.1145/383952.383955>
- [3] Steinberger, J. and Jezek, K. 2004. Using Latent Semantic Analysis in Text Summarization and Summary Evaluation. *Proceedings of ISIM ’04*, page 93-100.
- [4] David M Blei, Andrew Y Ng, and Michael I Jordan. 2003. Latent dirichlet allocation. the Journal of machine Learning research 3 (2003), 993–1022
- [5] Mark Steyvers and Tom Griffiths. 2007. Probabilistic topic models. Handbook of latent semantic analysis 427, 7 (2007), 424–440.
- [6] A. Nenkova and L. Vanderwende, “The impact of frequency on summarization,” Microsoft Research, Tech. Rep., 2005
- [7] Günes Erkan and Dragomir R Radev. 2004. LexRank: Graph-based lexical centrality as salience in text summarization. J. Artif. Intell. Res.(JAIR) 22, 1 (2004), 457–479
- [8] Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In Text Summarization Branches Out: Proceedings of the ACL-04 Workshop. 74– 81
- [9] H. P. Luhn, “The automatic creation of literature abstracts,” IBM J. Res. Dev., vol. 2, no. 2, pp. 159–165, Apr. 1958. [Online]. Available: <http://dx.doi.org/10.1147/rd.22.0159>
- [10] R. Mihalcea and P. Tarau, “TextRank: Bringing order into texts,” in Proceedings of EMNLP-04 and the 2004 Conference on Empirical Methods in Natural Language Processing, July 2004.
- [11] Nenkova and L. Vanderwende, “The impact of frequency on summarization,” Microsoft Research, Tech. Rep., 2005.
- [12] Julian S. Griggs, 2015. TL;DR: Automatic Summarization With Textual Annotations.
- [13] M. Belica. sumy. [Online]. Available: <https://github.com/miso-belica/sumy>
- [14] John M Conroy and Dianne P O’leary. 2001. Text summarization via hidden markov models. In Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval. ACM, 406–407.

- [15] Dou Shen, Jian-Tao Sun, Hua Li, Qiang Yang, and Zheng Chen. 2007. Document Summarization Using Conditional Random Fields.. In *IJCAI*, Vol. 7. 2862–2867.
- [16] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS)*.
- [17] Kristjan Arumae, Fei Liu. 2019. Guiding Extractive Summarization with Question-Answering Rewards.
- [23] Parth Mehta, Gaurav Arora, Prasenjit Majumder. 2018. Attention based Sentence Extraction from Scientific Articles using Pseudo-Labeled data. In *Proceedings of ACM Conference (Conference '17)*. ACM, New York, NY, USA, 4 pages
- [18] Jiwei Tan, Xiaojun Wan, and Jianguo Xiao. 2017. Abstractive document summarization with a graph based attentional neural model. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.
- [19] Logan Lebanoff, Kaiqiang Song, and Fei Liu. 2018. Adapting the neural encoder-decoder framework from single to multi-document summarization. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.
- [20] A. Nenkova and K. McKeown, “Automatic summarization,” *Foundations and Trends in Information Retrieval*, vol. 5, pp. 103–233, 2011.
- [22] S. Brin and L. Page. 1998. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7)
- [23] Mehdi Allahyari, Seyedamin Pouriyeh, Mehdi Assefi, Saeid Safaei, Elizabeth D. Trippe, Juan B. Gutierrez, and Krys Kochut. 2017. Text Summarization Techniques: A Brief Survey. In *Proceedings of arXiv, USA, July 2017*, 9 pages. <https://doi.org/10.1145/nnnnnnnn.nnnnnnnn>
- [24] Consumer Finance Protection Bureau. [Online]. Available: <https://www.consumerfinance.gov/data-research/consumer-complaints/>

8 How to Use My Project



The directory structure of my project is detailed above. Within the MyProject folder, there is **thesis.pdf**, which contains the project paper. It also contains **thesis_code_final.ipynb**, which is a jupyter notebook containing all the code for the tables made in this project. It does not, however, contain code for nonessential figures (such as the code for the various examples in the project). If you do not have jupyter notebook, the installation instructions can be found here: <https://jupyter.org/install>

The thesis folder also contains **LSAPlus_SumPlus.py**, which is the code described in the project paper. For more about the implementation of this code, refer to the project paper, under the Section 3.3. This file is referenced in **thesis_code_final.ipynb**. Lastly, there is **thesis_complaint_narratives_summaries.csv**, which is the CFPB dataset containing 50 rows.

Before running **thesis_code_final.ipynb**, be sure to install a few packages. Below are some quick ways to install most of the needed packages for python3. You may need to install a few more packages and python depending on your setup. **NOTE:** it could be “pip3 install” vs “pip install” for your computer depending on your setup and what version of python you have.

- pip install Sumy (most of the models are from Sumy, not from mysumy)
- pip install nltk (from terminal or from notebook) → from a cell in the jupyter notebook → import nltk → nltk.download('punkt') → nltk.download() → download all possible files (<https://www.nltk.org/install.html>)
- pip install Pandas
- pip install py-rouge (<https://pypi.org/project/py-rouge/>)
- pip install Numpy

Lastly, there is the mysumy file, which is a huge folder containing all of the original code for the original Sumy. There are two differences between mysumy (my implementation) and sumy, and they are:

- SumBasic algorithm has been adjusted to not update probabilities (this file is in: **MyProject/mysumy/sumy/summarizers/sum_basic.py**)
- I added *LSAPlus_SumPlus*, a function used in this project has been added (this file is in: **MyProject/mysumy/sumy/summarizers/LSAPlus_SumPlus**)

When you are ready to run the Jupyter Notebook:

1. Go to your terminal
2. Type in “jupyter notebook,” which will open the application on a browser in the thesis directory
3. Click on **thesis_code_final.ipynb**; be sure to have all needed packages installed
4. Once in **thesis_code_final.ipynb**, go to the “Cell” tab on the top
5. Click “Run All”, this will produce all of the relevant tables used in the project (keep in mind this can take a while as the code needs to fit a lot of models)
6. Scroll down to match the labelled tables in the code with the tables in the project paper
7. Try running the notebook just once, if you wish to run again, refresh the page and hit “Run All” as there may be some variables that share the same name