4-2022

Extractive Text Summarization Using Machine Learning

Swapnil Acharya

Extractive Text Summarization Using Machine Learning

by

Swapnil Acharya

A Starred Paper

Submitted to the Graduate Faculty of

Saint Cloud State University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in Computer Science

May, 2022

Abstract

We routinely encounter too much information in the form of social media posts, blogs, news articles, research papers, and other formats. This represents an infeasible quantity of information to process, even for selecting a more manageable subset. The process of condensing a large amount of text data into a shorter form that still conveys the important ideas of the original document is *text summarization*. Text summarization is an active subfield of *natural language processing*. *Extractive* text summarization identifies and concatenates important sections of a document sections to form a shorter document that summarizes the contents of the original document. We discuss, implement, and compare several *unsupervised machine learning* algorithms including *latent semantic analysis, latent dirichlet allocation*, and k-*means clustering*. *ROUGE-N* metric was used to evaluate summaries generated by these machine learning algorithms. Summaries generated by using *tf-idf* as a feature extraction scheme and *latent semantic analysis* had the highest *ROUGE-N* scores. This computer-level assessment was validated using an empirical analysis survey.

Acknowledgment

I would like to sincerely thank my advisor Dr. Maninder Singh, Department of Computer Science, and Information Technology (CSIT) for his suggestions and continuous mentorship throughout the research and execution of its implementation.

I would also like to express my gratitude to my committee member, Dr. Andrew Anda, Department of CSIT for his dedicated help in elevating several aspects of documentation of this research.

I would also like to thank my committee member Dr. Aleksandar Tomovic, Department of CSIT for his constructive feedback and support throughout the research. I would also like to thank Mr. Clifford Moran, Department of CSIT for his continuous and timely help in setting up classes, registration, and scheduling research presentations.

I would also like to thank all the members of the CSIT faculty at St. Cloud State University whose knowledge and expertise helped me to shape my academic career. I am very grateful for all my friends who helped me during the empirical analysis stage of this research. I would like to thank my family for their eternal support.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

Overview

We routinely encounter too much information in the form of social media posts, blogs, news articles, research papers, and other formats. This represents an infeasible quantity of information to process, even for selecting a more manageable subset. The process of computationally pruning any document into a shorter version in a manner that preserves as much information as possible and still conveys the overall idea of the original document is Text Summarization. *Text summarization* is an active subfield of *natural language processing* (NLP). Two popular approaches used today to generate automatic text summarization are *extractive text summarization* and *abstractive text summarization*. In *extractive text summarization*, based on their statistical and linguistic features, sentences or paragraphs are extracted from a source document and concatenated (Moratanch and Chitrakala 1). Whereas in *abstractive text summarization*, advanced NLP techniques are used to understand the source text and generate a new shorter text that conveys the most pertinent information of the source text (Dalal and Malik 1).

Objective

The objective of this research is to implement an *extractive text summarization* system. This system when given a large input text data will generate an extractive summary.

Chapter 2: Background Research

Overview

This chapter provides background on the approaches that use *machine learning* algorithms to generate extractive text summaries. *Artificial intelligence* (AI) is a branch of computer science that deals with creating intelligent machines that can think, behave and be able to make decisions on their own like human beings. (Kumar 118). *Machine learning* (ML) is a subfield of AI that enables a system to learn from data without being explicitly programmed (IBM Cloud Education). There are several categories of ML algorithms such as *supervised learning, unsupervised learning,* and *reinforcement learning*. In *supervised learning*, there is an established set of data and a certain understanding of how that data is classified (IBM Cloud Education). An iterative process, where data is analyzed without human supervision or intervention is known as *unsupervised learning* (IBM Cloud Education). In *reinforcement learning* algorithms, a user is guided to the best outcome via data analysis or trial and error process (IBM Cloud Education). This research will primarily focus on generating summaries via *unsupervised* ML algorithms.

Feature Engineering

Feature extraction is one of the essential parts of all NLP systems. ML algorithms need numerical input, however, the data in this research is textual. To process text data using ML algorithms, transformations that create numerical representations of the given text data must be applied. Three widely schemes used to create such transformations are bag of words (BOW), *term frequency-inverse document frequency* (TF-IDF), and *word-embeddings*. A group of sentences or documents is usually called a corpus in the field of NLP. BOW approach creates a matrix where the rows represent the sentences in a corpus and the columns represent the frequency of words in that corpus. For example, consider the following corpus *Apple is good*, *Apple is apple*, *Apple is healthy*. The BOW representation for this corpus is shown in Table 1.

Table 1

Bag Of Words Matrix

	Apple	Is	Good	Healthy
Sentence1	1	1	1	0
Sentence2	2	1	0	0
Sentence 3	1	1	0	1

Each row contains the frequency of words that appear in that sentence. However, the problem with the BOW scheme is that common words that appear frequently throughout the entire corpus have higher weights. For example, the word *apple* appears in all three sentences, this technically does not distinguish those three sentences from one another, however it has a higher weight. To distinguish between those three sentences words such as *good*, and *healthy* should be up-weighted because they are less frequent than other words. TF-IDF scheme combats this issue by emphasizing rare words and down-weighting frequent words. The mathematical formulas used by the TF-IDF scheme are shown by equations 1-3. Table 2 shows the resultant matrix after applying TF-IDF transformation.

$$TF = Frequency\ of\ a\ term\ in\ the\ current\ document \quad (1)$$

$$IDF = log(\frac{1 + Number\ of\ Documents}{1 + Frequency\ of\ a\ term\ in\ all\ Documents}) \quad (2)$$

$$TF - IDF = TF * IDF \quad (3)$$

Table 1

Term Frequency – Inverse Document Frequency Matrix

	Apple	Is	Good	Healthy
Sentence1	-0.0969	0.0000	0.3010	0.0000
Sentence2	-0.1938	0.0000	0.0000	0.0000
Sentence 3	-0.0969	0.0000	0.0000	0.3010

Table 2 shows that weights of rarely appearing words such as *good* and *healthy* are higher compared to weights of frequently appearing words such as *apple*. In trying to distinguish the effect of words, the TF-IDF scheme outperforms BOW scheme.

Word-embedding approach allows words with similar meanings to have similar representations. Individual words are represented as a real-value vector in a predefined vector space using this scheme. At the core of this algorithm, each word is mapped to one vector and the values are learned using neural networks. Jain et al. have used per sentence *word-embeddings* to generate features for their text summarization project (3). However, *word–embeddings* have high computational requirements compared to the previously mentioned approaches. Doing sentence-level word embeddings would require even more time and computation resources. This approach does not seem feasible, given the scale of this research. Padmakumar et al. have used sentence-level word-embeddings in their research, however, their research uses *supervised machine learning* models with labeled datasets (2).

Summarization using Latent Semantic Analysis

Latent semantic analysis (LSA) is an unsupervised ML algorithm. LSA applies statistical computation to a corpus of text whereby extracting and representing the contextual usage of words (Landauer et al. 2). LSA uses a method known as *singular value decomposition* (SVD) to decompose a single matrix into 3 different matrices. In LSA, a value *k*, meaning the number of *concepts* that will be kept must be specified. Then a matrix *X* of size *m*n* will be decomposed into 3 different matrices as shown by equation 4.

$$X = U * \sigma * V^T \qquad (4)$$

In equation 4, U will be an *m*k* matrix whose elements row-wise indicate documents, and the columns indicate *concepts* (Padmakumar et al. 3). σ will be a *k*k* diagonal matrix, whose elements represent the amount of variation captured from each *concept* (Padmakumar et al. 3). V^T will be a *k*n* matrix whose elements row-wise will be *concepts* and columns will be words (Padmakumar et al. 3). Padmakumar et al. use the TF-IDF scheme to generate a feature matrix, then use LSA to decompose that matrix, and finally use the decomposed matrices to generate scores for all sentences in the given corpus using equation 5 (2) . The sentences with the highest scores are selected for the extractive text summary.

$$Score\ (S_i) = \sqrt{\sum_{j=0}^{k} \sigma_j^2 \ * U_i^2} \qquad (5)$$

Summarization using Latent Dirichlet Allocation

Blei et al. proposed *latent dirichlet allocation* (LDA) in 2003 as an improvement over LSA (2). Unlike LSA, which is based on linear algebra, LDA is based on a probabilistic model. According to Blei et al., LDA is a three-level hierarchical Bayesian model, in which each item of a collection is modeled as a finite mixture over an underlying set of topics (1). Each topic is, in turn, modeled as an infinite mixture over an underlying set of topic probabilities. In the context of text modeling, the topic probabilities provide an explicit representation of a document. Like LSA, LDA assumes *k concepts* in a corpus, where these *k concepts* are distributed randomly i.e., each word in the corpus is assigned a random *concept*. LDA then assumes that each word in the corpus is assigned the wrong *concept* whereas other words are assigned correct *concepts*. Then based on a probabilistic model, this step is repeated. That probabilistic model is based on the number of *concepts* that exist in the corpus and the number of times a word had been assigned to a particular *concept* across the entire corpus. The iterative assignment stops when the *concept* reassignment process converges. Widjanarko et al. use LDA to generate an *extractive* text summary in their research (2). In their research, using LDA, *concepts* were identified, and sentences were selected for summary using the output of that LDA model. The basic idea in their paper is that they compute the probability of the sentence from the probabilities of words and *concepts*.

Summarization using *k*-means Clustering

One of the most widely used clustering algorithms, *k*-means uses distance measures to partition the data set into clusters (IBM Cloud Education). A cluster is a group of data that share a similarity. *k*-means is widely used as an *unsupervised* machine learning algorithm to categorize unlabeled data into groups/clusters represented by *k*. This algorithm partitions an input data set of size *n* into *k* separate groups, each described by the mean of data points in individual groups which are called *centroids*. This algorithm aims to choose centroids that attempt to minimize the inter-cluster sum of the squared criterion. Akter et al. use both LSA and *k*-means to generate automated text summaries (3). However, in this research both LSA and LDA are used for dimensionality reduction and then used as inputs to *k*-means clustering.

Other Approaches

Moratanch and Chitrakala discuss *supervised* machine learning algorithms for *text summarization* (2). In their paper, they train classification-based machine learning algorithms by providing summary and non-summary sentences, then the trained ML model predicts whether a sentence is a summary or non-summary sentence. They also discuss *unsupervised* ML approaches such as the *graph-based* similarity approach where linear algebraic models determine the importance of a sentence (2). Then a graph is built whose nodes represent the sentences in the corpus and edges represent cosine similarity values. Chen and Zhuge have used *recurrent neural networks* to generate extractive text summaries in their research (2).

Summary Evaluation

The performance of the *extractive* text summarization system must be assessed. Traditionally, ML models have been evaluated using metrics such as *accuracy, precision, recall*, and *f-measure*. Visually, these metrics are all part of the *confusion matrix*. For instance, consider a corpus where each document has been labeled as being summary or *non-summary*. This dataset is then used to train a *supervised* ML algorithm. This trained model can now classify a document being either *summary* or *non-summary*. This model is an example of binary classification. Table 3 shows an example of a confusion matrix that is used to evaluate a binary classifier model like this. In Table 3, *true positives* are those documents that were labeled as *summary* by humans and classified as *summary* by a ML model as well. *False positives* are those documents that were labeled as *non-summary* by humans but classified as *summary* by the ML model. *False negatives* are those documents that were labeled as *summary* but classified as *non-summary* by the ML model. *True negatives* are those documents that were labeled as *non-summary* by humans and classified as *non-summary* by the ML model as well.

Table 2

Binary Confusion Matrix

	Human Selected as *summary*	Human Selected as *non-summary*
ML model Selected as *summary*	True Positives (TP)	False Positive (FP)
ML model Selected as *non-summary*	False Negatives (FN)	True Negatives (TN)

The *accuracy* of an ML model indicates how often the model is correct. Equation 6 shows the formula used to calculate accuracy. The precision of an ML model indicates how many identifications were correct. Equation 7 shows the formula used to calculate the precision. Recall of an ML model indicates how many actual positives were identified correctly. Equation 8 shows the formula used to calculate recall. *F-measure* is the weighted average, or the harmonic mean of Precision and Recall. F-measure is typically used when a dataset is imbalanced. Equation 9 shows the formula used to calculate F-measure.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{6}$$

$$Precision = \frac{TP}{TP+FP} \tag{7}$$

$$Recall = \frac{TP}{TP+FN} \tag{8}$$

$$F - measure = \frac{2TP}{2TP+FP+FN} \tag{9}$$

These metrics have remained salient, especially in *supervised* ML algorithms used for classification. This is because the data used in *supervised* ML algorithms are labeled. However, *unsupervised* ML algorithms do not have labeled data, thus establishing evaluation metrics for these types of algorithms can be difficult. One of the popular metrics used today to evaluate automated text summaries is the ROUGE (Recall-Oriented Understudy for Gisting Evaluation) score. According to Lin, the ROUGE score includes measures to automatically determine the quality of a summary by comparing it to other (ideal) summaries created by humans (1). ROUGE measures the frequency of intersecting units such as *n*-gram, word sequences, and word pairs between the human-generated summary and the summary generated by computers (Lin 1). ROUGE-N gives *n*-gram recall between a computer generate summary and a reference summary generated by humans (Lin 1). Equation 10 shows the formula used to calculate ROUGE-N (Lin 1).

$$ROUGE-N = \frac{\sum_{S\,\in\{ReferenceSummaries\}} \sum_{gram_n \in\, S} Count_{match}(gram)_n}{\sum_{S\,\in\{ReferenceSummaries\}} \sum_{gram_n \in\, S} Count(gram)_n} \qquad (10)$$

In equation 10, n stands for the length of *n*-gram, $(gram)_n$ is the maximum number of *n*-grams co-occurring in the computer-generated summary and human-generated (reference) summary. Intuitively, in equation 10, the numerator is a count of *true positives* whereas the denominator is the count of *true positives* and *false negatives*. Therefore, Lin, describes ROUGE-N as being a recall-like measure since the denominator of the equation is the total sum of the number of *n*-grams in the human-generated summary (1). The downside of this evaluation metric is that the reference summary must be human-generated. During ML model tuning phase, for every corpus used in the evaluation of this model a reference summary must be created.

Chapter 3: Motivation

Overview

Application of text summarization ranges from scientific research, books, digest, stock market, and news to sporting events. During the background research phase of a research paper, normally abstracts of various reference articles are read for selection. However, if this research is to include the latest innovations and technologies, reading abstracts of multiple papers can be very tedious. *Text summarization* systems can be used to group multiple reference articles and generate a condensed abstract. Even teachers might not be able to frequently include recent technological trends in their teaching content. However, text summarizers can be used to generate a short document that conveys important ideas from multiple documents, which can then be easily included in classroom contents. In addition to that, as human beings, we might have some preconceived bias toward certain approaches, methods, topics, etc. However, the summarization algorithms are much less biased than human beings.

Chapter 4: Experiment Design

Overview

This chapter of the paper describes the steps proposed to create an automated *text summarization* system. As shown in Figure 1, first, an entire document (in pdf, word document, or other formats) will be read. Second, the text in the read document might contain unnecessary information which will be removed as part of text preprocessing. Third, using various mathematical transformations, relevant features will be extracted from the preprocessed text. Fourth, the extracted features will be used to train unsupervised ML algorithms to create ML models. Finally, the output of the ML models will be used to generate a summary of the document.

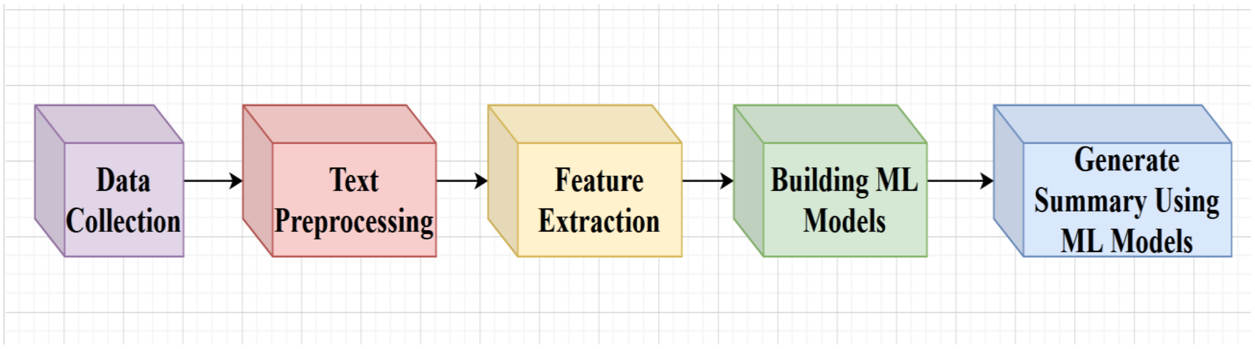


Figure 1. Proposed steps involved in creating an Automatic Text Summarization system

Research Questions

In addition to the methods discussed in background research, there might be other approaches used to generate better *extractive* summaries. The following research questions will be investigated along with the execution of the experiment design.

a) *RQ-1:* What metrics exist to score semantic sentence similarities?

   LSA uses SVD to decompose a matrix into 3 different matrices. How can these 3 matrices be used to define a mathematical model that can be used to generate a score for each document in the input corpus?

b) *RQ-2:* How do *n*-grams impact *extractive* summary generation?

   During the feature extraction process, how does the co-occurrence range of words affect the outcome of the final summary? Are trigrams better than bigrams or a combination of all trigrams, bigrams, and unigrams must be used when generating BOW and TF-IDF weights?

c) *RQ-3:* How effective is the *word-embeddings* approach in generating features?

   BOW and especially TF-IDF remain popular feature generation schemes in the field of NLP. However, rather than using these classical approaches, can *word-embeddings* scheme be used? Are the summaries generated using *word-embeddings* better than the ones generated by BOW and TF-IDF?

Overall Design

The overall experiment design is depicted in the image below. Upcoming sections describe each step-in detail.

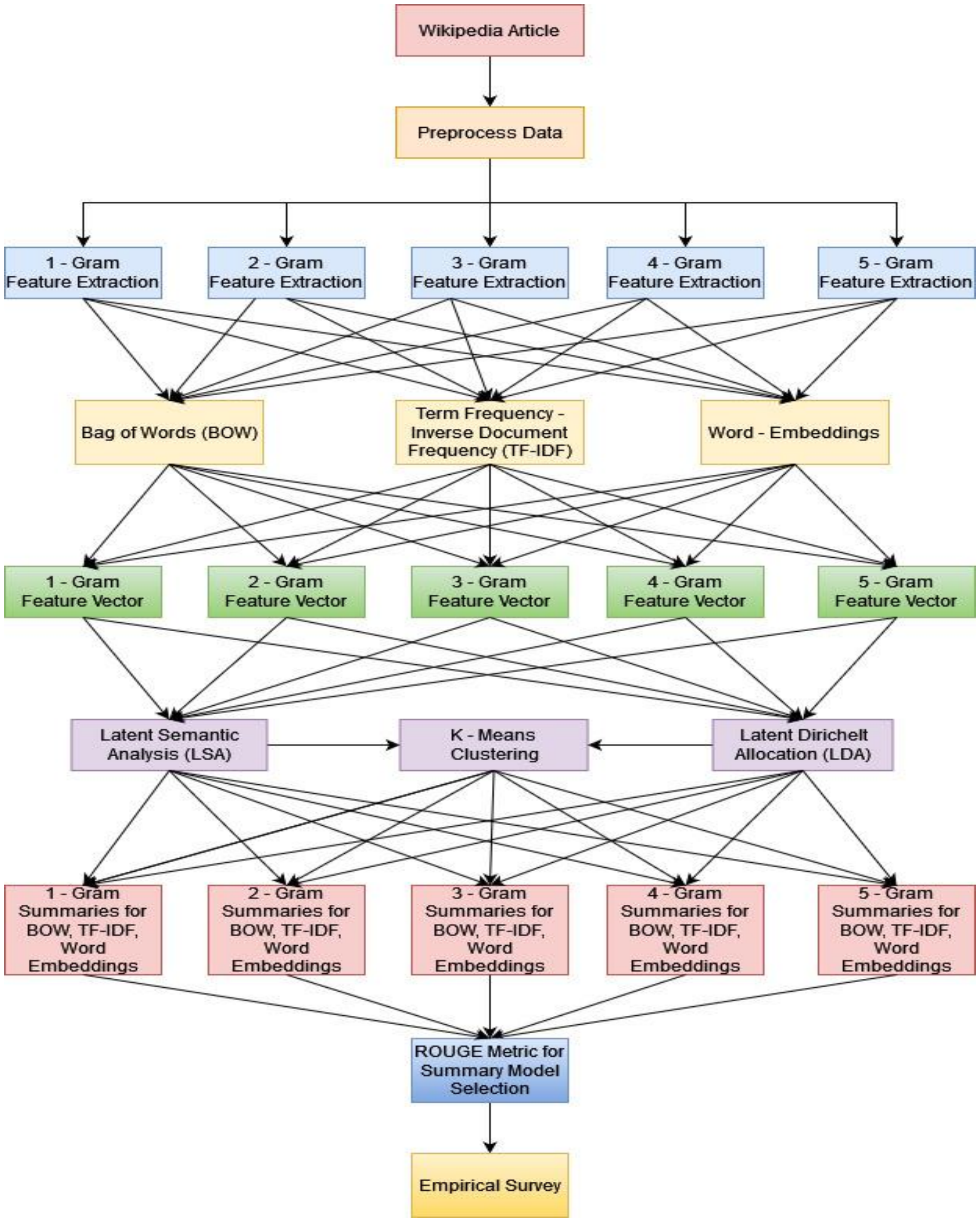

Wikipedia Article
Preprocess Data
1 - Gram Feature Extraction
2 - Gram Feature Extraction
3 - Gram Feature Extraction
4 - Gram Feature Extraction
5 - Gram Feature Extraction
Bag of Words (BOW)
Term Frequency - Inverse Document Frequency (TF-IDF)
Word - Embeddings
1 - Gram Feature Vector
2 - Gram Feature Vector
3 - Gram Feature Vector
4 - Gram Feature Vector
5 - Gram Feature Vector
Latent Semantic Analysis (LSA)
K - Means Clustering
Latent Dirichelt Allocation (LDA)
1 - Gram Summaries for BOW, TF-IDF, Word Embeddings
2 - Gram Summaries for BOW, TF-IDF, Word Embeddings
3 - Gram Summaries for BOW, TF-IDF, Word Embeddings
4 - Gram Summaries for BOW, TF-IDF, Word Embeddings
5 - Gram Summaries for BOW, TF-IDF, Word Embeddings
ROUGE Metric for Summary Model Selection
Empirical Survey

Figure 2. Overall system design

Data Collection

The data used in this research could be lab reports, research papers, and various essay-based assignments. An ideal corpus could be built by collecting research papers from various academic disciplines. One source could be the university's graduate school online repository. However, for the given scale of research, *Wikipedia articles* are sufficient to demonstrate the idea.

Text Preprocessing

Text preprocessing is a very integral part of any NLP system. The steps involved in the text preprocessing part are shown in Figure 2.

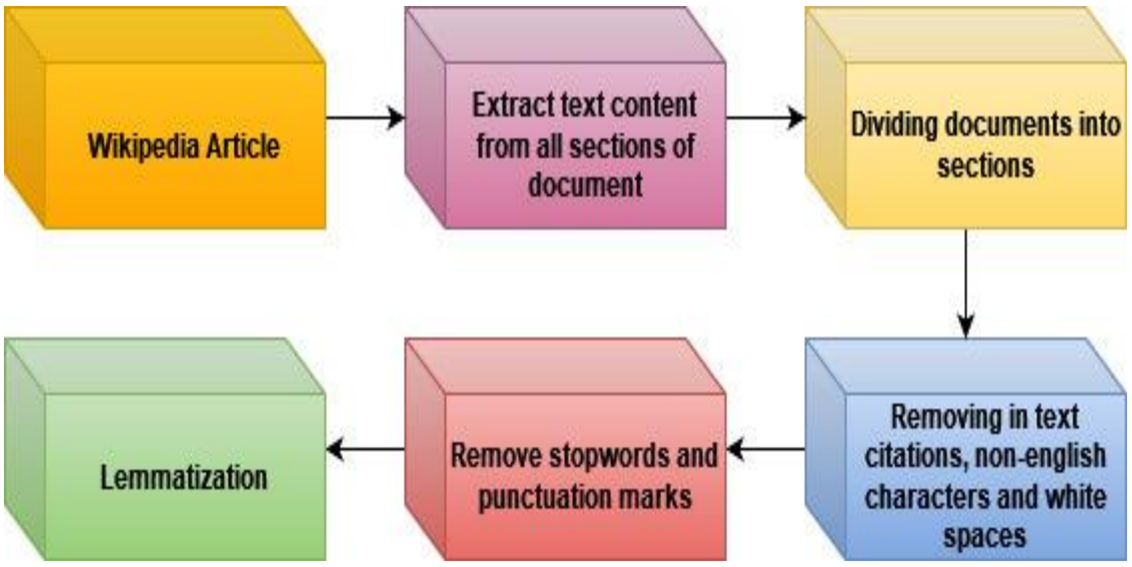

Wikipedia Article
Extract text content from all sections of document
Dividing documents into sections
Removing in text citations, non-english characters and white spaces
Remove stopwords and punctuation marks
Lemmatization

Figure 3. Proposed steps involved in Text Preprocessing

a) Parsing HTML document (Web Scraping)

The text in Wikipedia articles is contained inside HTML tags such as <p></p> etc. Those textual data must be parsed from HTML tags and put into a single string as part of preprocessing. Since the data used in this research is textual, images in Wikipedia articles will not be used. Beautiful Soup is a python library that is used for web scraping purposes to pull data out of HTML and XML files.

b) Sentence Level Tokenization

Once the text from the Wikipedia article is extracted, the extracted text will be a single large string. This single large string should be divided into individual sentences. One way to accomplish this would be to break this large string on the *period* symbol delimiter. However, there might be occasional *period* symbols, in a single string, the previous approach would fail in this event. NLTK (Natural Language Toolkit), is a suite of libraries and programs for symbolic and statistical natural language processing written in Python programming language. NLTK provides means to divide a large corpus into text. This library is widely used in NLP and ML fields to accomplish a task.

c) Noise Removal

Wikipedia article might contain data that might skew results. This data or noise should be removed during preprocessing step. Some sources of noise include citation numbers, newline characters, whitespaces, and non-english characters.

d) Word Level Tokenization

At this stage, the data should be in the format of sentence tokens i.e., divided into individual sentences. However, for further preprocessing, each sentence must be broken down into individual words. One possible approach to accomplish this task would be to split a sentence on whitespace delimiter. However, the NLTK library that provided functionality for the earlier task of sentence-level tokenization also provides functionality for word-level tokenization. This word tokenization functionality will be used to accomplish this task.

e) Stopwords and Punctuation marks removal

At this stage, the data should be in the format of sentence tokens and each sentence token should include word tokens. This step includes the removal of stop words and punction marks. Stopwords are simply a set of commonly used words in a language, which do not carry significant information. Examples of stop words are *a*, *the*, *is*, *are*, etc. In addition to removing stopwords, punctuation marks such as *!*, *?*, @, %, etc. must also be removed.

f) Lemmatization

The English language has several variants of a single term. For example, consider the word *change*, its variations include *changing*, *changes*, *changed,* etc. However, these words, for the most part, convey the same meaning. To preserve the meaning of words, reducing variants to their root form is highly desired. This task can be accomplished by either a process called *Stemming*, or another process called *Lemmatization*. *Stemming,* without consideration for a word’s meaning crudely chops suffix such as *ing*, *ed* from all words. For example, consider variants of *change*: changing, changes, and changed. *Stemming* operation will change all the variants to the word *chang*. *Lemmatization* on the other hand uses vocabulary and morphological analysis of words to reduce variants to root form. For example, consider variants of the *change*: changing, changes, and changed. A *lemmatization* operation will change the variants to the word *change*. *Stemming* reduced variants to *chang* whereas *lemmatization* reduced variants to *change*. The result of *lemmatization* is more desirable in this case because the reduced output itself, i.e., *change* has actual meaning compared to *chang* which does not have any meaning. Based on these results, it could be inferred that l*emmatization* should be preferred over *stemming*. However, *lemmatization* and *stemming* operations have tradeoffs. *Lemmatization* maintains a detailed dictionary to ensure correct mappings are produced; parsing through this dictionary takes a decent amount of time. *Stemming* on the other hand does not have any lookup operation, it simply chops off suffixes, hence *stemming* is faster than *lemmatization*. Therefore, *stemming* is widely used in scenarios where extremely large corpora are used. *Lemmatization* on the other hand is used on small corpora. This research uses a relatively small Wikipedia article, so *lemmatization* was considered over *stemming*.

Feature Extraction

As discussed earlier in background research, feature extraction schemes under consideration are BOW, TF-IDF, and word-embeddings. Before diving deep into the implementations, it is important to understand the meaning behind *n*-gram. While working with textual data, it is important to determine the meaning of words. For example, consider the following words *I*, *ate*, *apple*. Individually they have meaning but they might not convey the context of their meaning. However, those three words when grouped, not only convey the meaning of induvial words but provide context as well. Thus, during the feature extraction process, the selection of the number of contextual words is essential. If the meaning of the single word is considered then, it would indicate *n*=1 i.e., 1-gram features, similarly, if the meaning of two words is considered then, it would indicate *n*=2 i.e., 2-gram features, and so on. Consider the following corpus: *I love reading books*. Different *n*-gram features for this corpus would look like the following:

1-Gram: *I*, *love*, *reading*, *books*

2-Gram: *I love*, *love reading*, *reading books*

3-Gram: *I love reading*, *love reading books*

The *n*-gram range can be specified in the feature extraction scheme such as BOW, TF-IDF, and word-embeddings. *Scikit-learn* is a free software machine learning library for *python* programming language. It provides functionality for feature extraction schemes such as BOW and TF-IDF which allow users to specify the *n*-gram range as a parameter. As for *word-embeddings* we will use another library called *genism*. It is also an open-source library designed for NLP.

Building Machine Learning Models

As discussed earlier in background research, this research will heavily focus on using *unsupervised* ML algorithms. The generated/extracted features from the previous step are used to train machine learning algorithms like LSA, LDA, and *k*-means clustering. The outputs of these models are then used to select sentences for the summary. This section will discuss the inputs and outputs of these machine learning algorithms.

a) Latent Semantic Analysis (LSA)

   First a preprocessed Wikipedia article is used by three extraction schemes BOW, TF-IDF, and word-embeddings to generate *sentence-word* vectors. These *sentence-word* vectors are used to train an LSA algorithm. The trained LSA model then finally produces *sentence-topic*, *topic–topic* and *term-topic* vectors. These three vectors will be then used to select sentences for the summary. This process is repeated for all three different feature extraction schemes and for each extraction scheme, the process is repeated for 5 different *n*-gram ranges (1-5). Overall, this will produce 3 x 5 = 15 different summaries.

b) Latent Dirichlet Allocation (LDA)

   First the preprocessed Wikipedia article is used by 3 extraction schemes BOW, TF-IDF, and word-embeddings to generate *sentence-word* vectors. This vector is used to train the LDA model. The trained LDA model then finally produces *sentence-topic* and *term-topic* vectors. These two vectors will be then used to select sentences for the summary. This process is repeated for three different feature extraction schemes and for each extraction scheme, the process is repeated for 5 different *n*-gram ranges (1-5). Overall, this will produce 3 x 5 = 15 different summaries.

c) *k*-means Clustering

First the preprocessed Wikipedia article is used by 3 extraction schemes BOW, TF-IDF, and word-embeddings to generate *sentence-word* vectors. These *sentence-word* vectors are used to train the LSA model. The trained LSA model then finally produces *sentence-topic* vectors. This vector will be further used to train the *k*-means clustering model. The outputs of the *k*-means clustering model will be then used to select sentences for the summary. This process is repeated by replacing the LSA model with LDA. This process is repeated for three different feature extraction schemes and for each extraction scheme, the process is repeated for 5 different *n*-gram ranges (1-5). Overall, this will produce 2 x 3 x 5 = 30 different summaries.

Sentence Selection for Summary

The final step will be to select sentences using the outputs of trained ML models. This section will discuss how outputs of individual ML models are used to rank and select sentences for the summary.

a) Sentence selection using LSA

The previous steps describe the outputs of the LSA model, those outputs will be then used with mathematical equation 5 discussed in the background section to select sentences for the summary.

b) Sentence selection using LDA

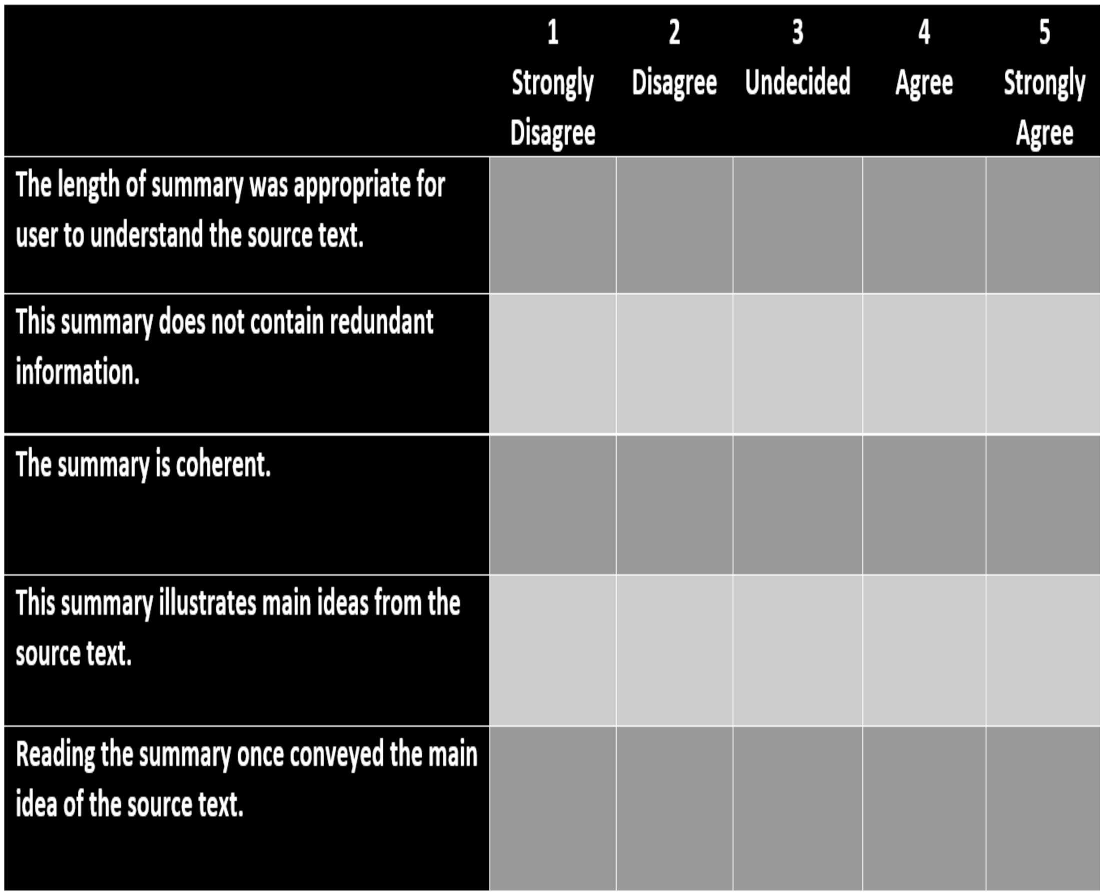
The outputs of LDA are two vectors, *sentence-topic*, and *topic-terms*. The sentence selection will be done using *sentence-topic* vectors. The algorithm will be to divide sentences into topics, based on the probability values of topics in each sentence vector. Once the sentences are grouped into topics, the sentence that has the highest probability within the topic group will be selected as the winner within the topic group. The winning sentences will be selected for the summary.

c) Sentence selection using *k*-means

The outputs of *k*-means model will be cluster centers, often called centroids. The sentence selection will be done using these centroids. Centroids represent the arithmetic mean of their clusters. In terms of sentences, centroids can be thought of as those sentences that represent the most important idea within their group of sentences. However, centroids themselves are calculated as the arithmetic mean amount groups of sentences, so they do not represent a precise sentence. The idea would be to pick the sentences that have the closest Euclidean distance to the centroids.

Evaluation

The final step will be to evaluate the generated summaries. This section will discuss summary evaluation methods. Generated summaries will be compared to a reference summary using ROUGE-N metric. Based on the outputs of ROUGE-N metrics, 3 high-scoring summaries will be selected for empirical analysis. The empirical analysis will be done using a survey form, which will be sent out to participants from diverse backgrounds. This form will have participants read a Wikipedia article about Hurricane Irene, then they will be presented with 3 selected summaries. The participants will read the source text, then read the selected summaries and finally evaluate them on some questionaries. The assessment will be done using Likert Scale as show by figure 4.



	1 Strongly Disagree	2 Disagree	3 Undecided	4 Agree	5 Strongly Agree
The length of summary was appropriate for user to understand the source text.					
This summary does not contain redundant information.					
The summary is coherent.					
This summary illustrates main ideas from the source text.					
Reading the summary once conveyed the main idea of the source text.					

Figure 4. Survey form Likert scale.

Chapter 5: Experiment Procedure

Overview

This chapter of the paper describes the steps proposed to create an automated summarization system. As shown in Figure 1, First, an entire Wikipedia Article will be read. Second, the text in the read document might contain unnecessary information which will be removed as part of text preprocessing. Third, using various mathematical transformations, relevant features will be extracted from the preprocessed text. Fourth, the extracted features will be used to train unsupervised ML algorithms to create ML models. Finally, the output of the ML models will be used to generate a summary of the document.

Algorithmic Steps for Research Questions

a) *RQ-1*: What metrics exist to score semantic sentence similarities?

LSA uses singular value decomposition (SVD) to decompose a matrix into 3 different matrices. How can these 3 matrices use to define a mathematical model that can be used to generate a score for each document in the input corpus?

Padmakumar et al., have used LSA to generate an extractive text summary (1). The algorithmic steps are as follows:

1. Generate feature matrix using TF-IDF feature extraction Scheme.
2. Decompose feature matrix (*X*) into three matrices. X = $U * \sigma * V^T,$ where U will be an *m*k* matrix whose elements row-wise indicate documents, and the columns indicate *concept*s. σ will be a *k*k* diagonal matrix, whose elements represent the amount of variation captured from each concept. V^T will be a *k*n* matrix whose elements row-wise will be *concepts* and columns will be words.

3. Use the decomposed matrices to score individual sentences using equation 5.
4. Once all the scores have been generated, select N top-scoring sentences in descending order.
5. Finally, concatenate the N sentences to form an extractive text summary.

b) *RQ-2:* How do N-grams impact extractive summary generation?

During the feature engineering process, how does the co-occurrence range of words affect the outcome of the final summary? Are trigrams better than bigrams or a combination of all trigrams, bigrams, and unigrams must be used when generating BOW and TF-IDF weights?

The general idea is to generate a feature matrix for 1-gram, 2-gram, 3 -gram, and combinations. The algorithmic steps are as follows:

1. Generate BOW and TF-IDF matrix for 1-gram, 2-gram, 3-gram, 4-gram, and 5-gram.
2. Generate summaries for all the feature matrices generated from Step 1.
3. Compare the summaries generated by these feature matrices using the ROUGE-N evaluation method.

c) *RQ-3:* How effective is the *word-embeddings* approach in generating features?

BOW and especially TF-IDF remain popular feature generation processes in the field of NLP. However, rather than using these classical approaches, can word embedding be used? Are the summaries generated via the use of word embeddings better than BOW and TF-IDF?

The general idea is to use the weights generated by the word-embeddings scheme rather than, BOW and TF-IDF scheme. The algorithm steps that will be taken to accomplish this task are as follows:

1. Generate word embeddings for the corpus.
2. Use the vocabulary generated by word embeddings as features.
3. Generate a matrix that holds the average for each word's embeddings, which are in the word embedding vocab.
4. Based on the features, develop a feature matrix where if a sentence contains the word in word embedding vocab, use the average value for that word from step 3 as the weight, else assign that word value of 0.
5. Generate summaries for all the feature matrices using steps discussed earlier in research question 1.

Implementation for Generating Summary Using LSA

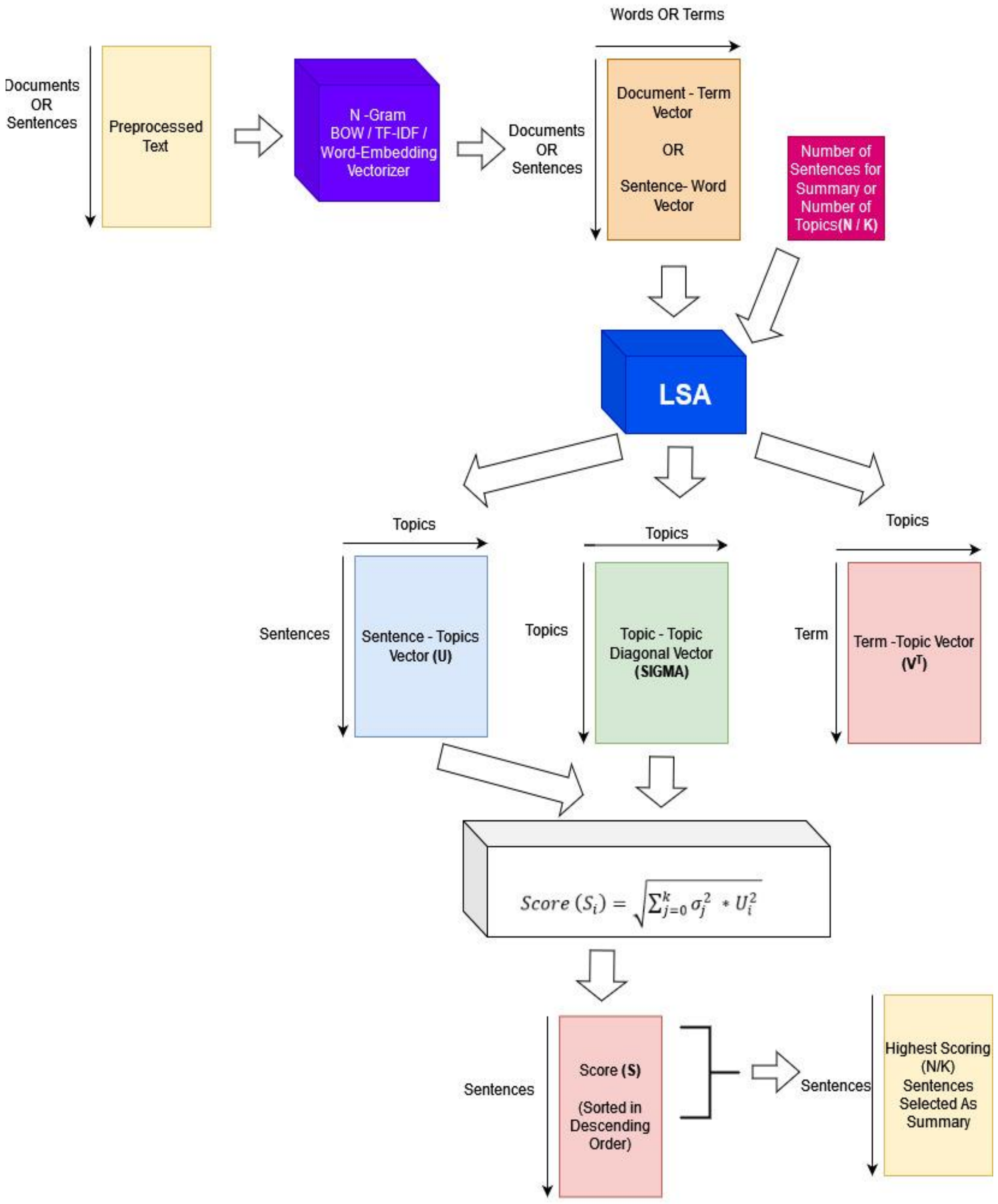

Documents OR Sentences
Preprocessed Text
N -Gram BOW / TF-IDF / Word-Embedding Vectorizer
Words OR Terms
Documents OR Sentences
Document - Term Vector OR Sentence- Word Vector
Number of Sentences for Summary or Number of Topics(N / K)
LSA
Topics
Sentences
Sentence - Topics Vector (U)
Topics
Topics
Topic - Topic Diagonal Vector (SIGMA)
Topics
Term
Term -Topic Vector (V^T)
$Score\ (S_i) = \sqrt{\sum_{j=0}^{k} \sigma_j^2 * U_i^2}$
Sentences
Score (S) (Sorted in Descending Order)
Sentences
Highest Scoring (N/K) Sentences Selected As Summary

Figure 5. System design for generating summary using LSA

The following algorithmic steps were taken to execute the experiment design:

1. Web scrape a Wikipedia Article using Beautiful Soup library, whereby extracting the text contents include <*p*>*…*</*p*> HTML tag.
2. Preprocess the scaped text by removing whitespace, references, non-english words, stop words, and lemmatizing the remaining words. This step heavily relies on regular expressions, word, and sentence tokenizers, and wordnet lemmatizer.
3. For exploratory data analysis, word clouds of raw and preprocessed text were generated using the *word-cloud* library. Figure 7 shows the code implementation of this step.
4. BOW, TF-IDF, and word-embeddings features were extracted for the 1,2,3,4,5-gram range using APIs provided by sci-kit learn and genism respectively.
5. Using LSA, feature matrices generated in step 4 were decomposed into 3 different matrices, *sentence-topic* (U), *topic-topic* (σ), and *term-topic* (V^T).
6. All sentences are scored using Equation 5, which uses the matrices from step 5.
7. Sentences with the highest scores are selected for the summary.
8. Wikipedia article about Hurricane Irene was sent out to a few colleges who then generated an extractive summary by reading the article.
9. Using the reference summaries from step 6, the ROUGE-N metric was used to evaluate the automated summary.

Implementation for Generating Summary Using LDA

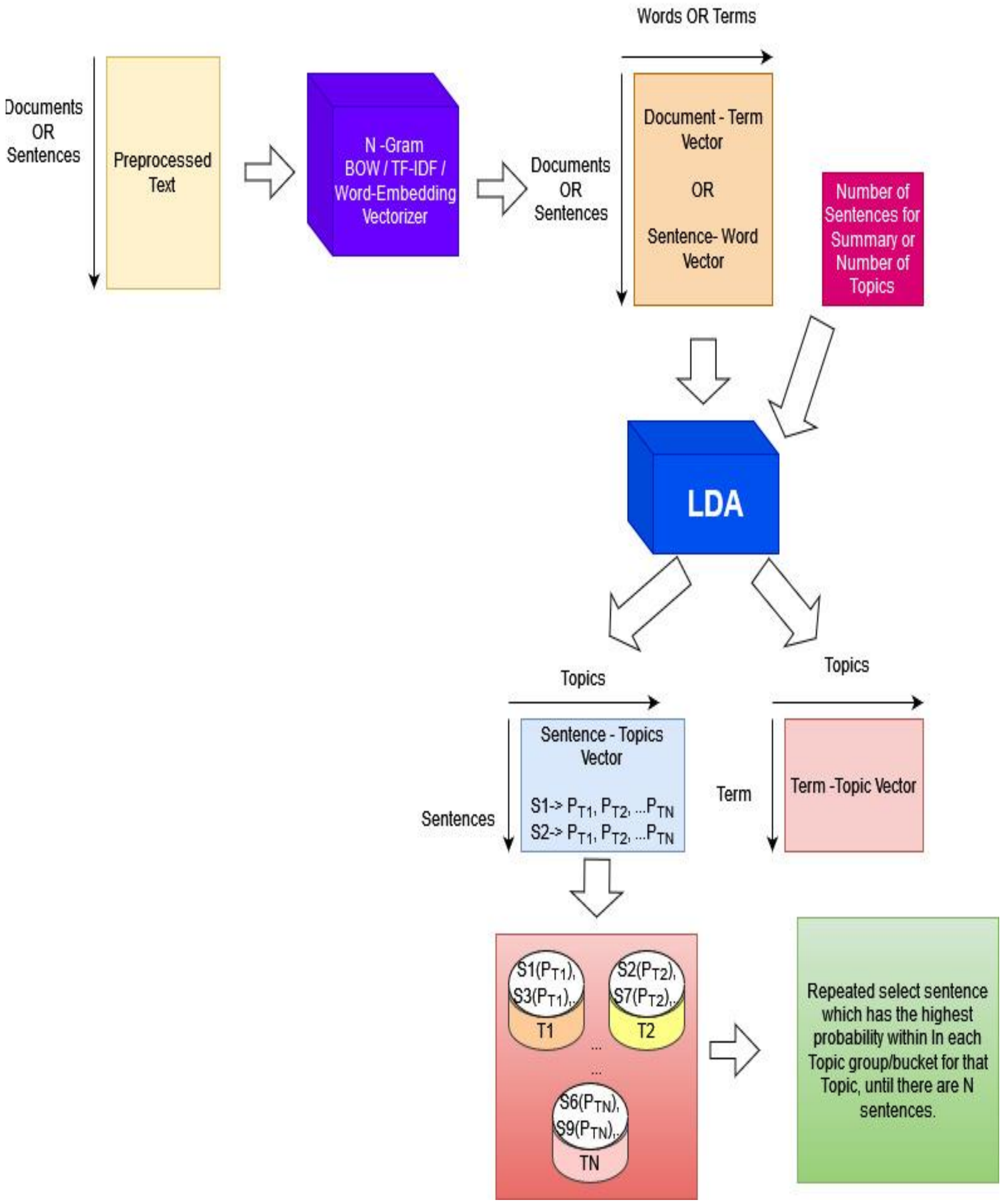

Documents OR Sentences
Preprocessed Text
N -Gram BOW / TF-IDF / Word-Embedding Vectorizer
Words OR Terms
Documents OR Sentences
Document - Term Vector OR Sentence- Word Vector
Number of Sentences for Summary or Number of Topics
LDA
Topics
Sentences
Sentence - Topics Vector
S1-> P_T1, P_T2, ...P_TN
S2-> P_T1, P_T2, ...P_TN
Topics
Term
Term -Topic Vector
S1(P_T1), S3(P_T1),.. T1
S2(P_T2), S7(P_T2),.. T2
S6(P_TN), S9(P_TN),.. TN
Repeated select sentence which has the highest probability within In each Topic group/bucket for that Topic, until there are N sentences.

Figure 6. System design for generating summary using LDA

The following algorithmic steps were taken to execute the experiment design:

1. Web scrape a Wikipedia Article using Beautiful Soup library, whereby extracting the text contents include <p>…</p> HTML tag.
2. Preprocess the scaped text by removing whitespace, references, non-english words, stop words, and lemmatizing the remaining words. This step heavily relied on regular expressions, word, and sentence tokenizers, and wordnet lemmatizer.
3. For exploratory data analysis, word clouds of raw and preprocessed text were generated using the *word-cloud* library. Figure 7 shows the code implementation of this step.
4. BOW, TF-IDF, and word-embedings features were extracted for the 1,2,3,4,5-gram range using APIs provided by sci-kit learn and genism respectively.
5. Using LDA, feature matrices generated in step 4 were used to train LDA model. The output of the trained LDA model was a *sentence-topic* vector.
6. Each sentence vector on *sentence-topic* matrix has probabilities for *k* Topics, a sentence would be assigned to the topic which has the highest probability value in that sentence vector. This was repeated for all sentences in *sentence-topic* matrix.
7. Once the sentences were grouped by topics, sentences with the highest probability among each topic group were selected for the summary.
8. Wikipedia article about Hurricane Irene was sent out to a few colleges who then generated an extractive summary by reading the article.
9. Using the reference summaries from step 6, the ROUGE-N metric was used to evaluate the automated summary.

Implementation for Generating Summary Using K-means

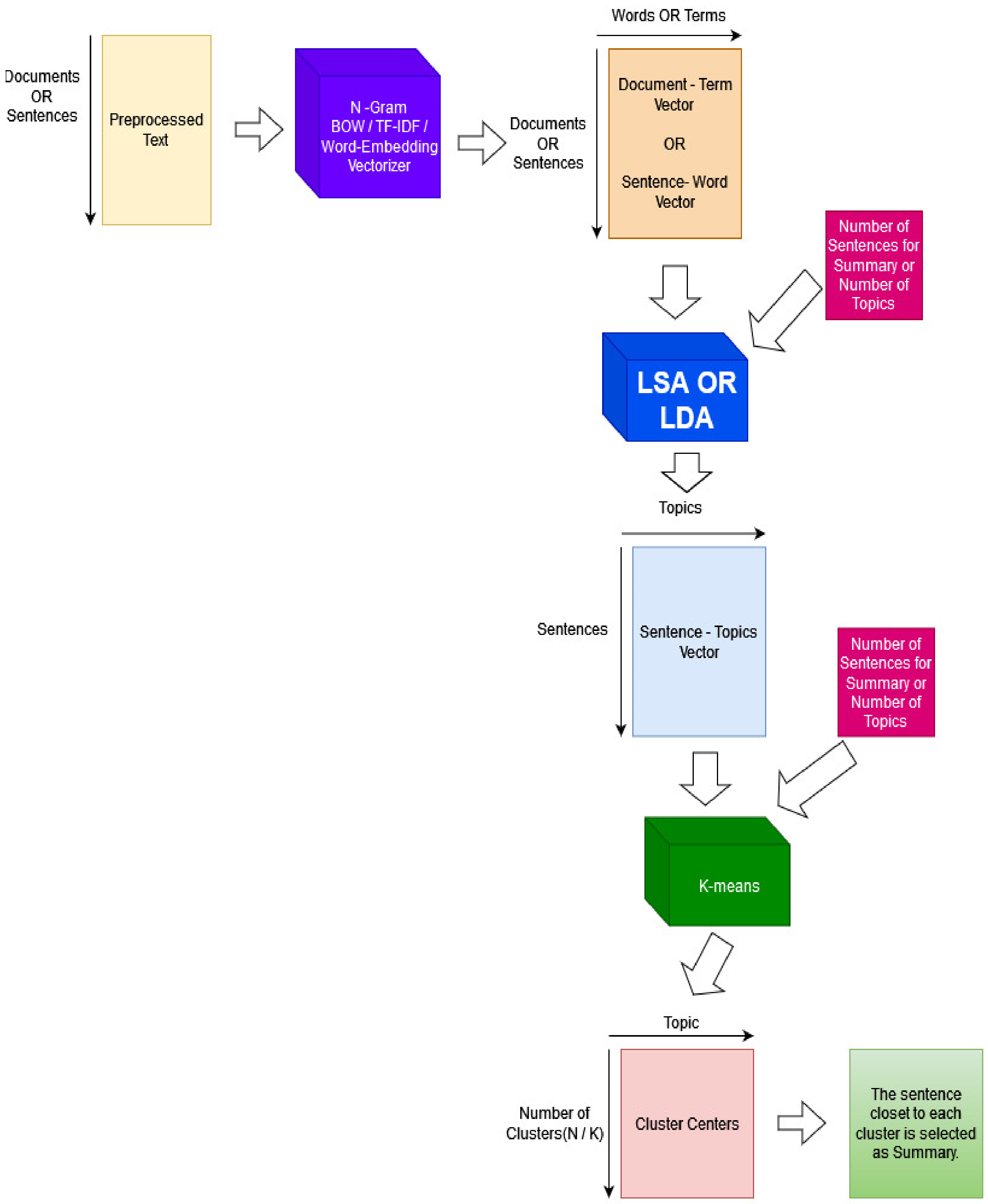

Documents OR Sentences
Preprocessed Text
N -Gram BOW / TF-IDF / Word-Embedding Vectorizer
Words OR Terms
Documents OR Sentences
Document - Term Vector OR Sentence- Word Vector
Number of Sentences for Summary or Number of Topics
LSA OR LDA
Topics
Sentences
Sentence - Topics Vector
Number of Sentences for Summary or Number of Topics
K-means
Topic
Number of Clusters(N / K)
Cluster Centers
The sentence closet to each cluster is selected as Summary.

Figure 7. System design for generating summary using K-means

The following algorithmic steps were taken to execute the experiment design:

1. Web scrape a Wikipedia Article using Beautiful Soup library, whereby extracting the text contents include <*p*>*…*</*p*> HTML tag.
2. Preprocess the scaped text by removing whitespace, references, non-English words, stop words, and lemmatizing the remaining words. This step heavily relied on regular expressions, word, and sentence tokenizers, and wordnet lemmatizer.
3. For exploratory data analysis, word clouds of raw and preprocessed text were generated using the *word-cloud* library. Figure 7 shows the code implementation of this step.
4. BOW, TF-IDF, and word-embeddings features were extracted for the 1,2,3,4,5-gram range using APIs provided by sci-kit learn and genism respectively.
5. Using LSA/LDA, feature matrices generated in step 4 were used to train LDA model. The output of the trained LDA model is a *sentence-topic* matrix.
6. The *sentence-topic* vector was then used to train *k*-means clustering algorithm.
7. The outputs of the trained *k*-means model were cluster centers, formally known as *centroids*.
8. Distance between a centroid and all vectors in *sentence-topic* matrix were calculated.
9. The *sentence-topic* vector with the smallest distance would be selected as a summary.
10. Steps 8-9 were repeated for all centroids. This would eventually give one sentence per cluster. These sentences were then used to form the final summary.
11. Wikipedia article about Hurricane Irene was sent out to a few colleges who then generated an extractive summary by reading the article.
12. Using the reference summaries from step 6, the ROUGE-N metric was used to evaluate the automated summary.

Chapter 6: Results and Discussions

Overview

This chapter will list the results of the experiment procedure and then will further analyze those results.

Results

a) Word Cloud: Word cloud is widely used in the analysis of textual data. Typically, word clouds are used for exploratory data analysis purposes. The purpose of word-cloud in this research was just to ensure that unwanted data such as HTML tags do not go undetected during the preprocessing stage.

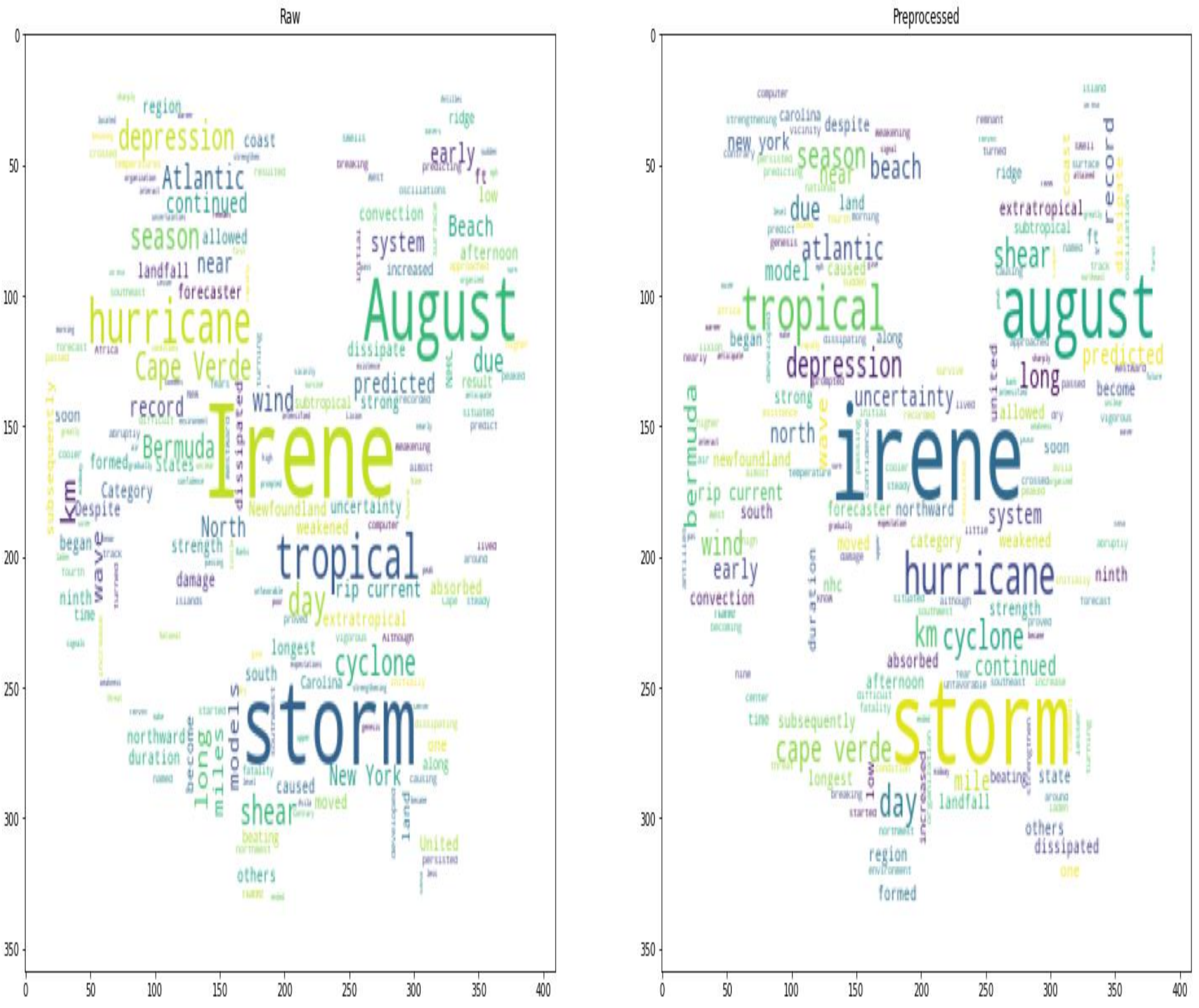


Figure 8. Raw (left) and Preprocessed (right) Word clouds of article about Hurricane Irene.

b) Generated Summaries

In this research a total of, the following methods were used to generate summaries:

i) BOW and LSA
ii) TF-IDF and LSA
iii) Word-Embeddings and LSA
iv) BOW and LDA
v) TF-IDF and LDA
vi) Word-Embeddings and LDA
vii) BOW, LSA, and K-means.
viii) TF-IDF, LSA, and K-means.
ix) Word-Embeddings, LSA, and K-means.
x) BOW, LDA, and K-means.
xi) TF-IDF, LDA, and K-means.
xii) Word-Embeddings, LDA, and K-means.

In this research a total of 12 methods were used to generate summaries, each of those methods was repeated for the *n*-gram range from 1to 5. An overall total of 12x5=60 summaries were generated. Rather than listing all the summaries, this section contains the evaluation of those summaries when compared to a human-generated summary using ROUGE-N F-measure scores.

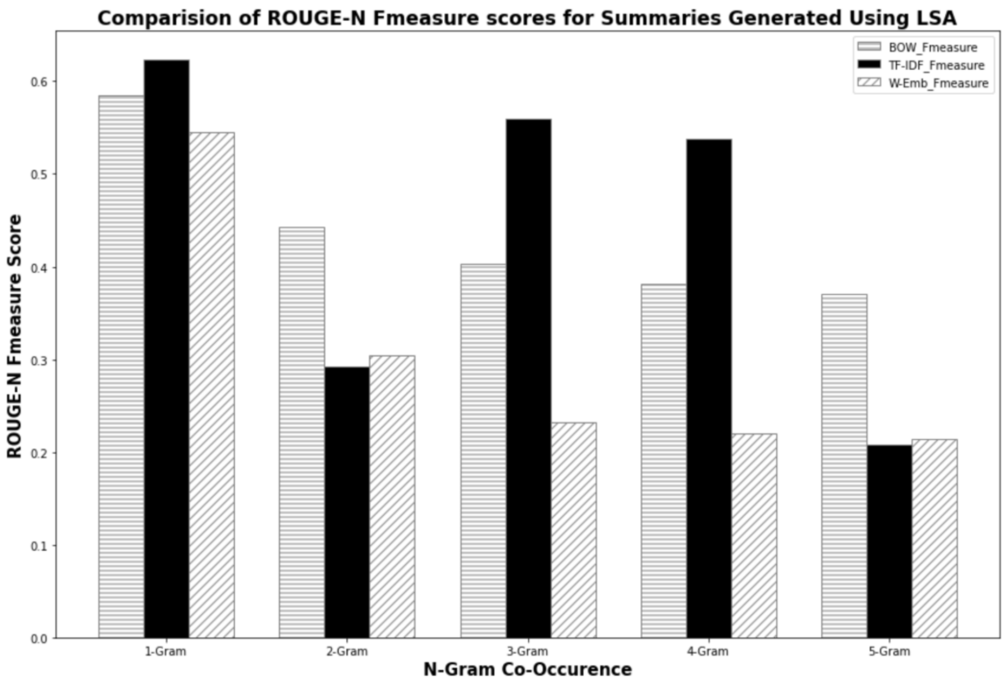


Figure 9. ROUGE-N F-measure scores for summaries generated using LSA.

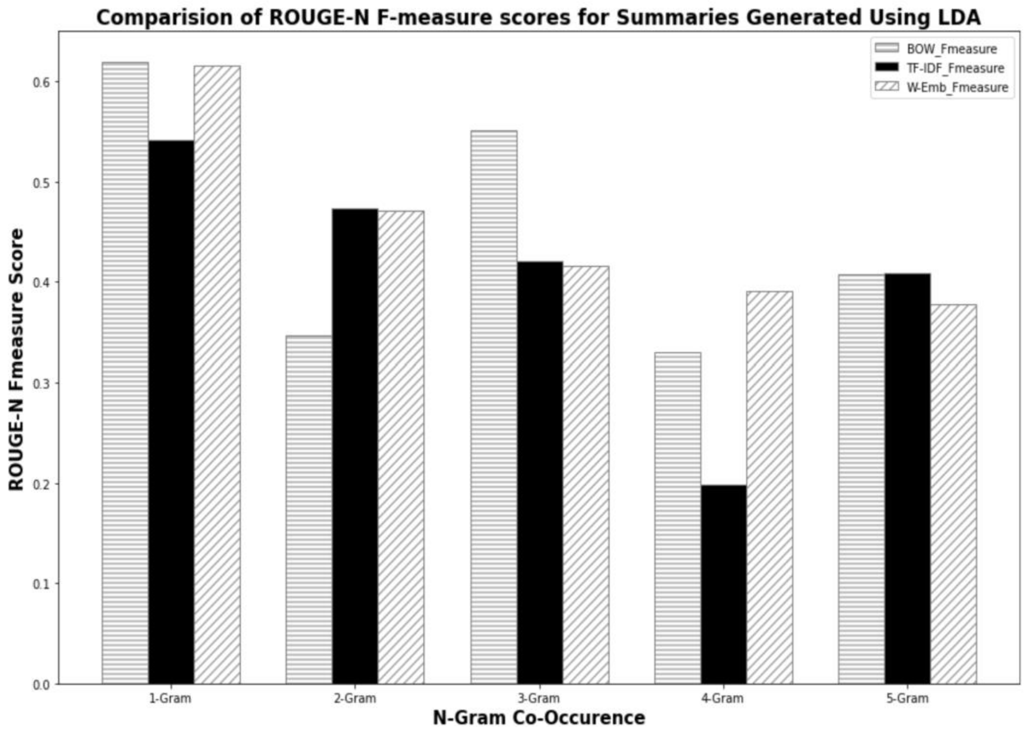


Figure 10. ROUGE-N F-measure scores for summaries generated using LDA.

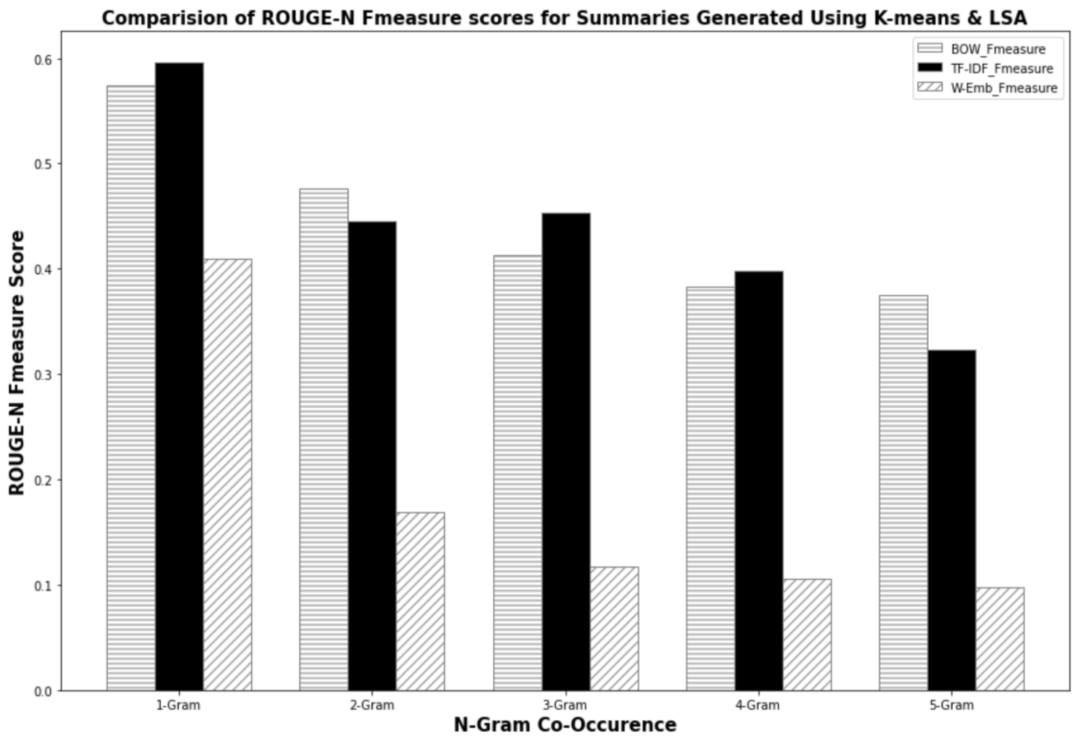


Figure 11. ROUGE-N F-measure scores for summaries generated using LSA and K-means.

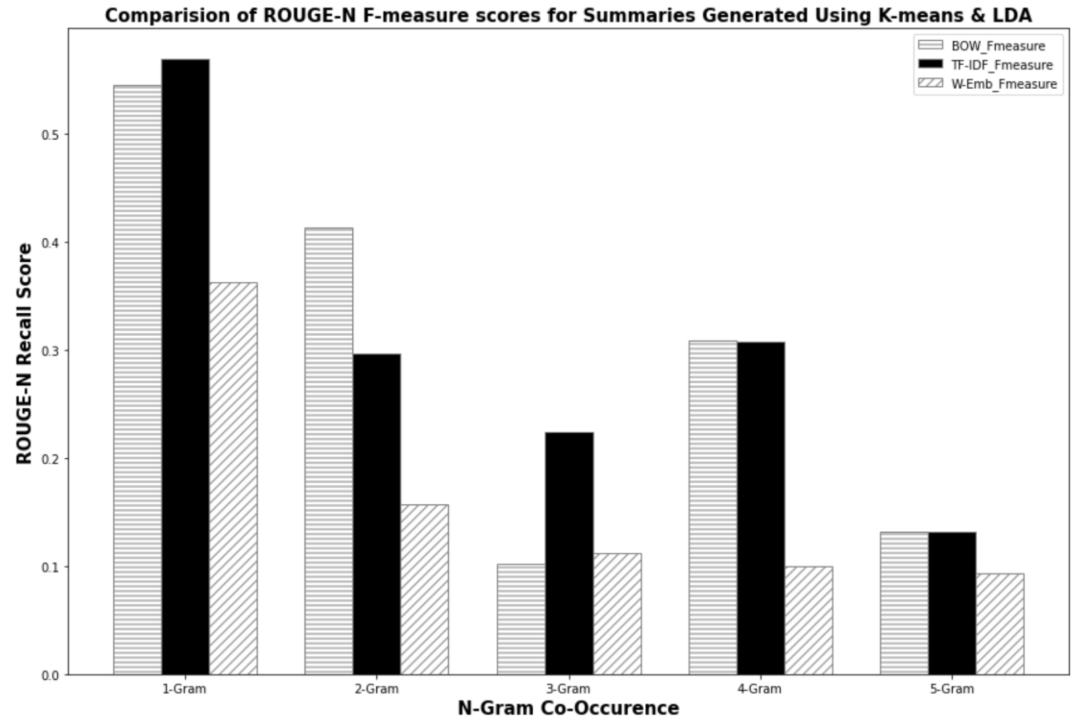


Figure 12. ROUGE-N F-measure scores for summaries generated using LDA and K-means.

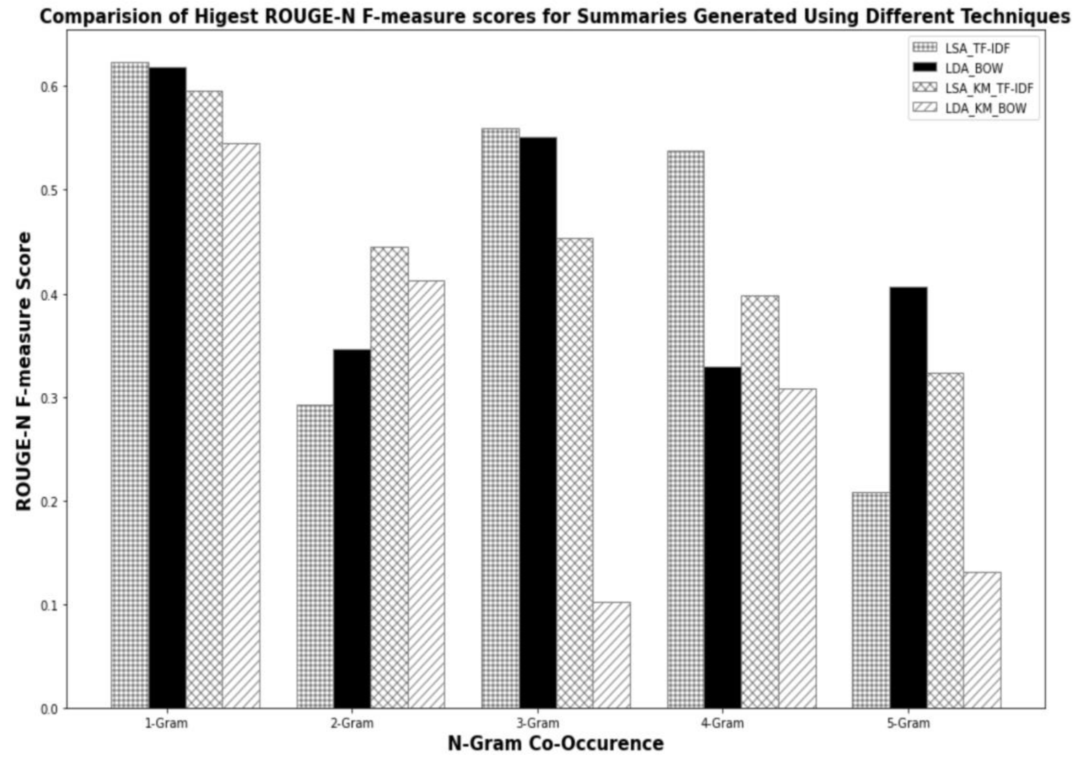
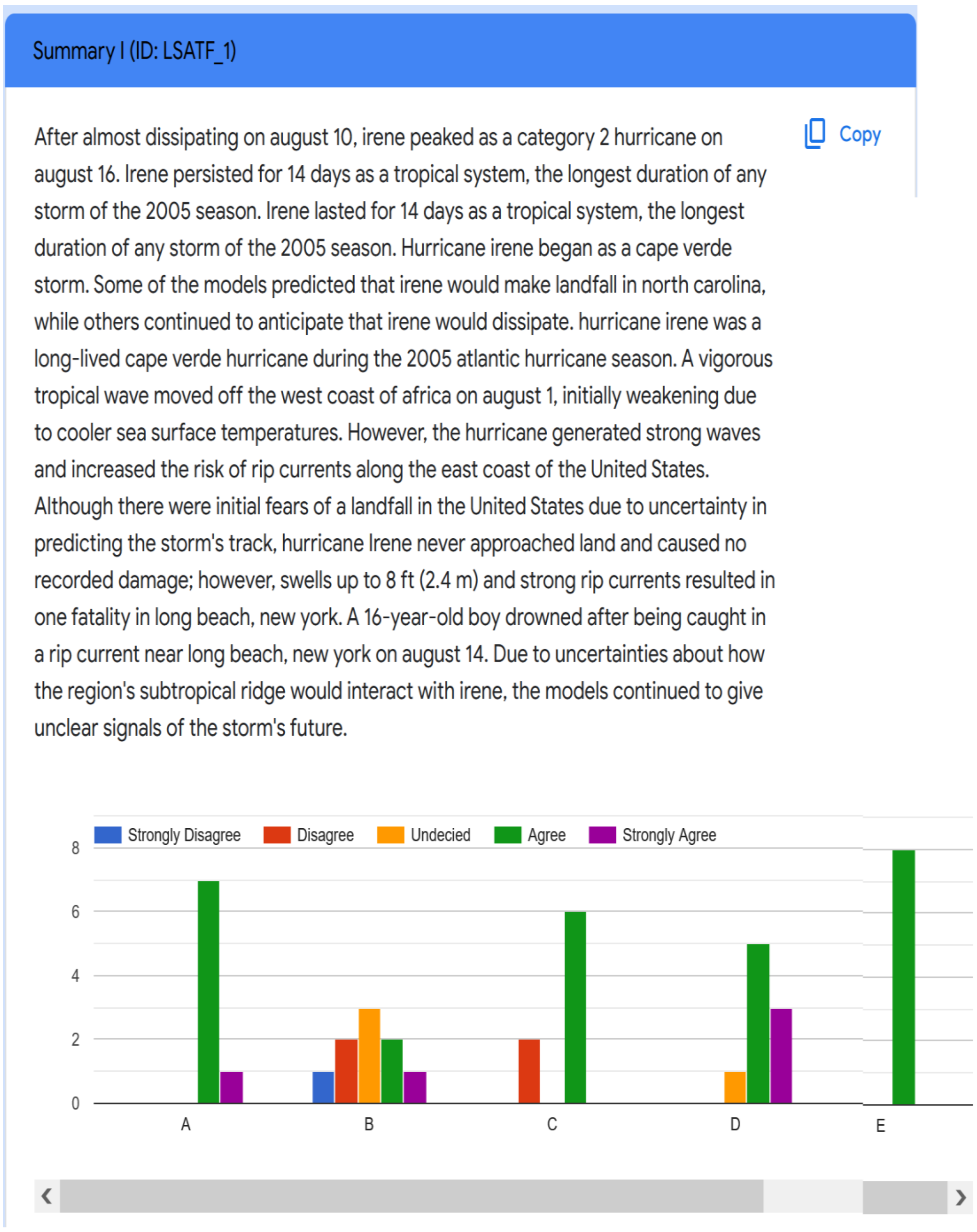


Figure 13. ROUGE-N F-measure scores among all generated summaries.

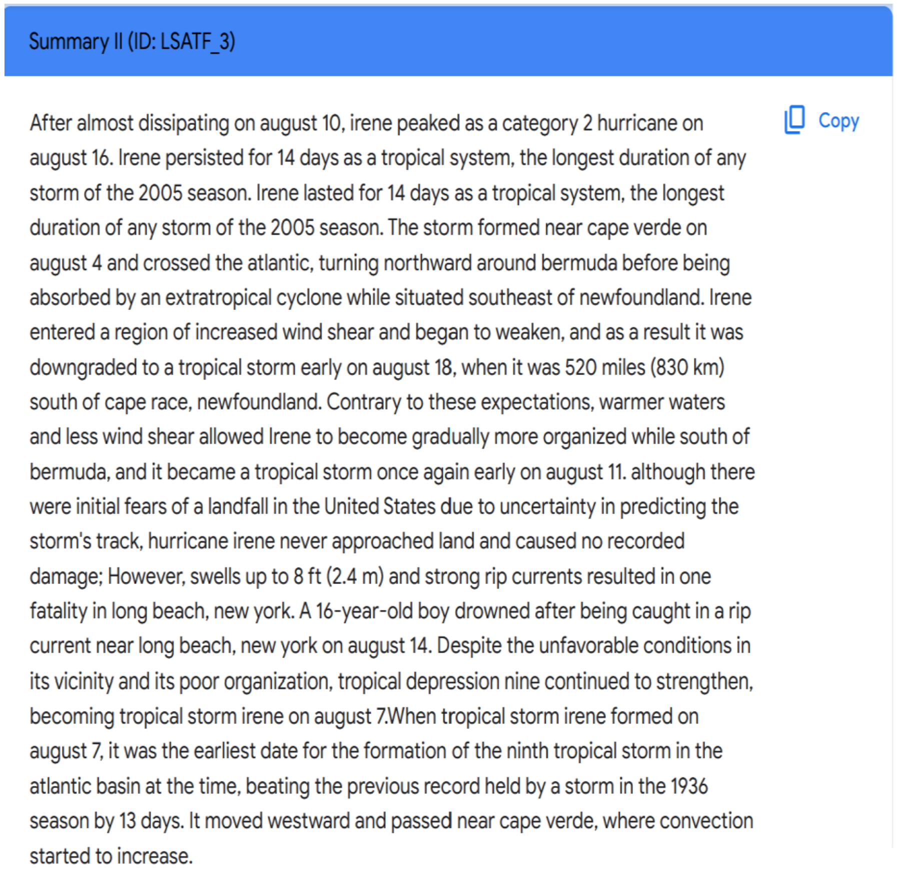


Summary I (ID: LSATF_1)

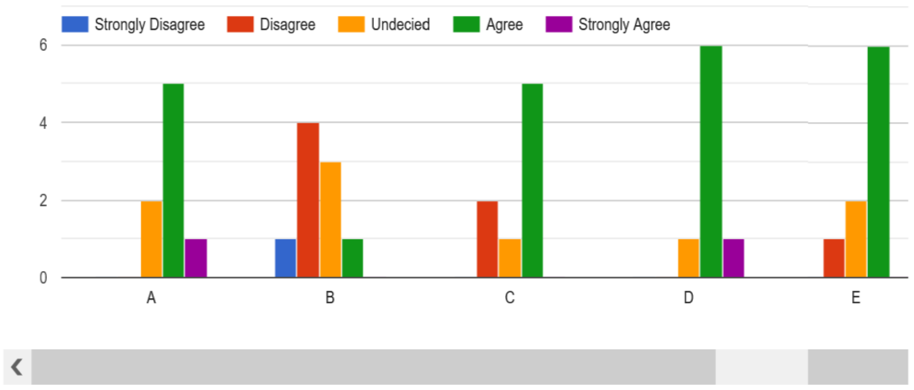
After almost dissipating on august 10, irene peaked as a category 2 hurricane on august 16. Irene persisted for 14 days as a tropical system, the longest duration of any storm of the 2005 season. Irene lasted for 14 days as a tropical system, the longest duration of any storm of the 2005 season. Hurricane irene began as a cape verde storm. Some of the models predicted that irene would make landfall in north carolina, while others continued to anticipate that irene would dissipate. hurricane irene was a long-lived cape verde hurricane during the 2005 atlantic hurricane season. A vigorous tropical wave moved off the west coast of africa on august 1, initially weakening due to cooler sea surface temperatures. However, the hurricane generated strong waves and increased the risk of rip currents along the east coast of the United States. Although there were initial fears of a landfall in the United States due to uncertainty in predicting the storm's track, hurricane Irene never approached land and caused no recorded damage; however, swells up to 8 ft (2.4 m) and strong rip currents resulted in one fatality in long beach, new york. A 16-year-old boy drowned after being caught in a rip current near long beach, new york on august 14. Due to uncertainties about how the region's subtropical ridge would interact with irene, the models continued to give unclear signals of the storm's future.

A: The length of the summary was appropriate for user to understand source text.
B: This summary does not contain redundant information.
C: The summary is coherent.
D: This summary illlustrates main ideas from the source text.
E: Reading the summary once converyed the main idea of source text.

Figure 14. Empirical Analysis for summary generated using 1-gram TF-IDF and LSA.

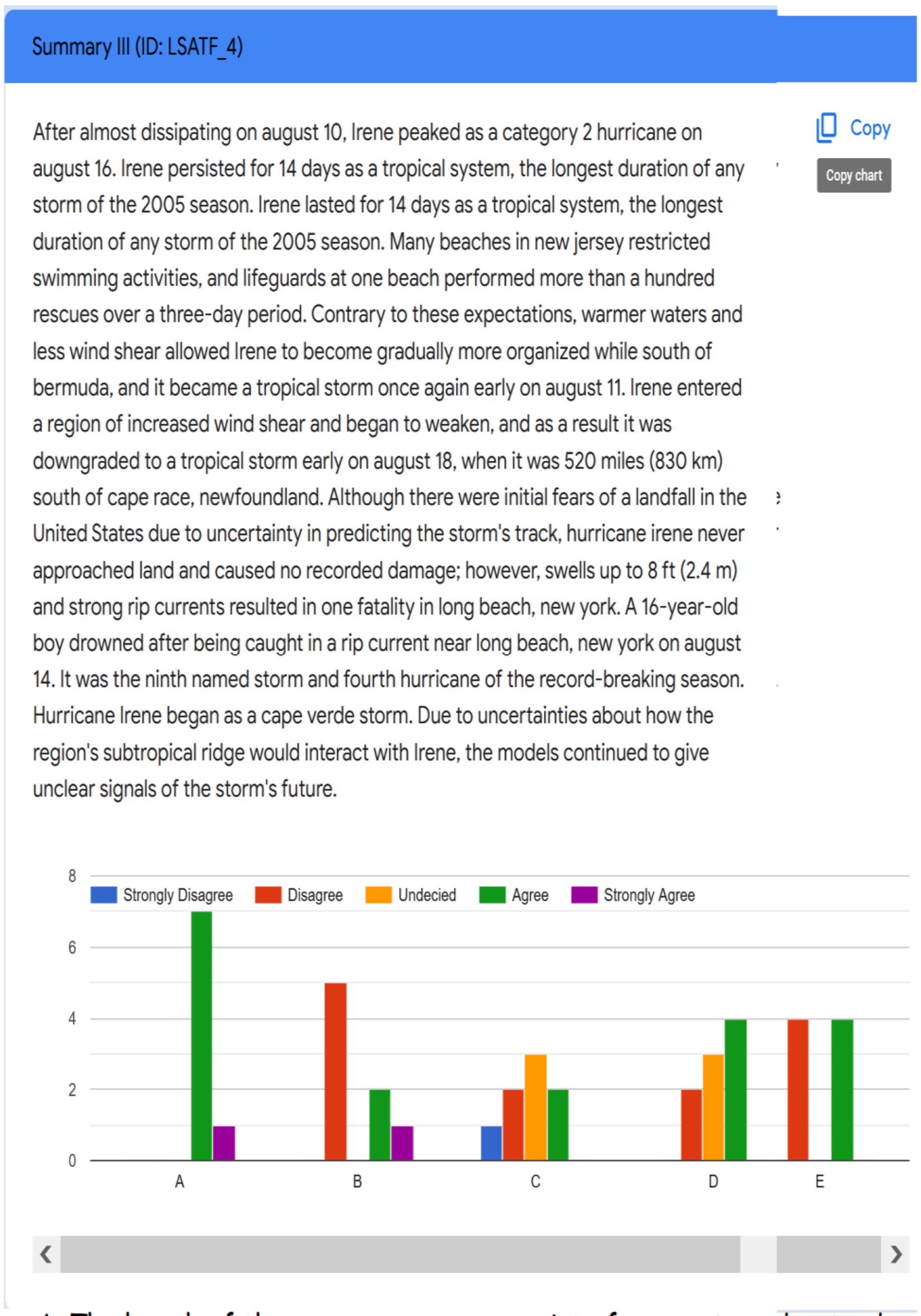

Summary II (ID: LSATF_3)

After almost dissipating on august 10, irene peaked as a category 2 hurricane on august 16. Irene persisted for 14 days as a tropical system, the longest duration of any storm of the 2005 season. Irene lasted for 14 days as a tropical system, the longest duration of any storm of the 2005 season. The storm formed near cape verde on august 4 and crossed the atlantic, turning northward around bermuda before being absorbed by an extratropical cyclone while situated southeast of newfoundland. Irene entered a region of increased wind shear and began to weaken, and as a result it was downgraded to a tropical storm early on august 18, when it was 520 miles (830 km) south of cape race, newfoundland. Contrary to these expectations, warmer waters and less wind shear allowed Irene to become gradually more organized while south of bermuda, and it became a tropical storm once again early on august 11. although there were initial fears of a landfall in the United States due to uncertainty in predicting the storm's track, hurricane irene never approached land and caused no recorded damage; However, swells up to 8 ft (2.4 m) and strong rip currents resulted in one fatality in long beach, new york. A 16-year-old boy drowned after being caught in a rip current near long beach, new york on august 14. Despite the unfavorable conditions in its vicinity and its poor organization, tropical depression nine continued to strengthen, becoming tropical storm irene on august 7.When tropical storm irene formed on august 7, it was the earliest date for the formation of the ninth tropical storm in the atlantic basin at the time, beating the previous record held by a storm in the 1936 season by 13 days. It moved westward and passed near cape verde, where convection started to increase.



A: The length of the summary was appropriate for user to understand source text.
B: This summary does not contain redundant information.
C: The summary is coherent.
D: This summary illlustrates main ideas from the source text.
E: Reading the summary once converyed the main idea of source text.

Figure 15. Empirical Analysis for summary generated using 3-gram TF-IDF and LSA.

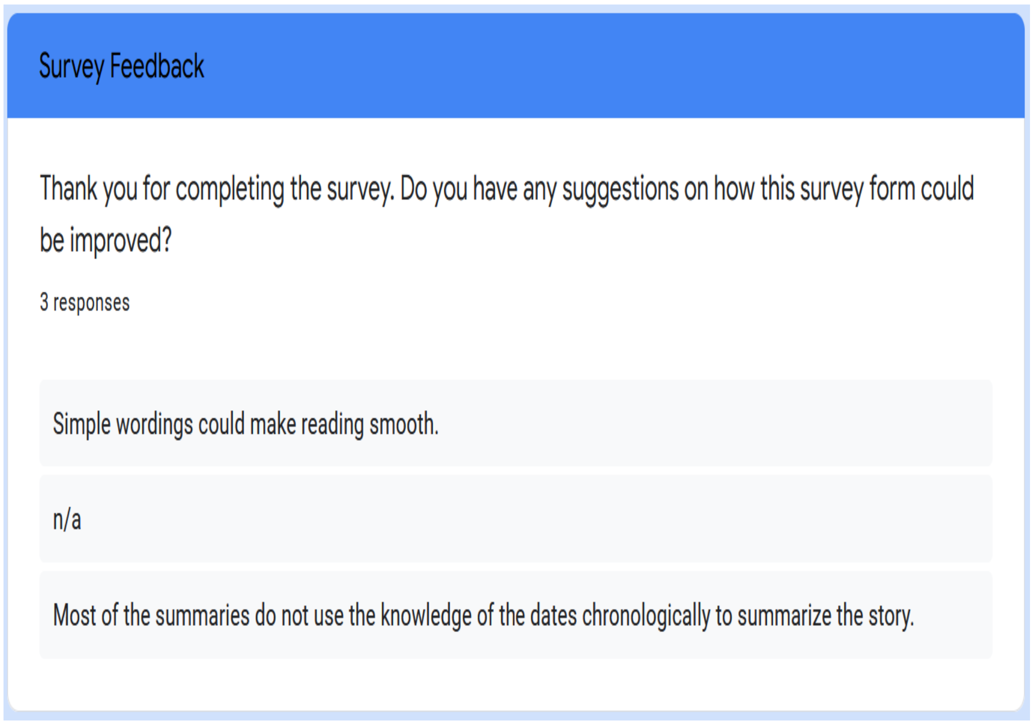


Summary III (ID: LSATF_4)

After almost dissipating on august 10, Irene peaked as a category 2 hurricane on august 16. Irene persisted for 14 days as a tropical system, the longest duration of any storm of the 2005 season. Irene lasted for 14 days as a tropical system, the longest duration of any storm of the 2005 season. Many beaches in new jersey restricted swimming activities, and lifeguards at one beach performed more than a hundred rescues over a three-day period. Contrary to these expectations, warmer waters and less wind shear allowed Irene to become gradually more organized while south of bermuda, and it became a tropical storm once again early on august 11. Irene entered a region of increased wind shear and began to weaken, and as a result it was downgraded to a tropical storm early on august 18, when it was 520 miles (830 km) south of cape race, newfoundland. Although there were initial fears of a landfall in the United States due to uncertainty in predicting the storm's track, hurricane irene never approached land and caused no recorded damage; however, swells up to 8 ft (2.4 m) and strong rip currents resulted in one fatality in long beach, new york. A 16-year-old boy drowned after being caught in a rip current near long beach, new york on august 14. It was the ninth named storm and fourth hurricane of the record-breaking season. Hurricane Irene began as a cape verde storm. Due to uncertainties about how the region's subtropical ridge would interact with Irene, the models continued to give unclear signals of the storm's future.

A: The length of the summary was appropriate for user to understand source text.
B: This summary does not contain redundant information.
C: The summary is coherent.
D: This summary illlustrates main ideas from the source text.
E: Reading the summary once converyed the main idea of source text.

Figure 16. Empirical Analysis for summary generated using 3-gram TF-IDF and LSA.



Survey Feedback

Thank you for completing the survey. Do you have any suggestions on how this survey form could be improved?

3 responses

Simple wordings could make reading smooth.

n/a

Most of the summaries do not use the knowledge of the dates chronologically to summarize the story.

Figure 17. Feedback from Empirical Analysis candidates

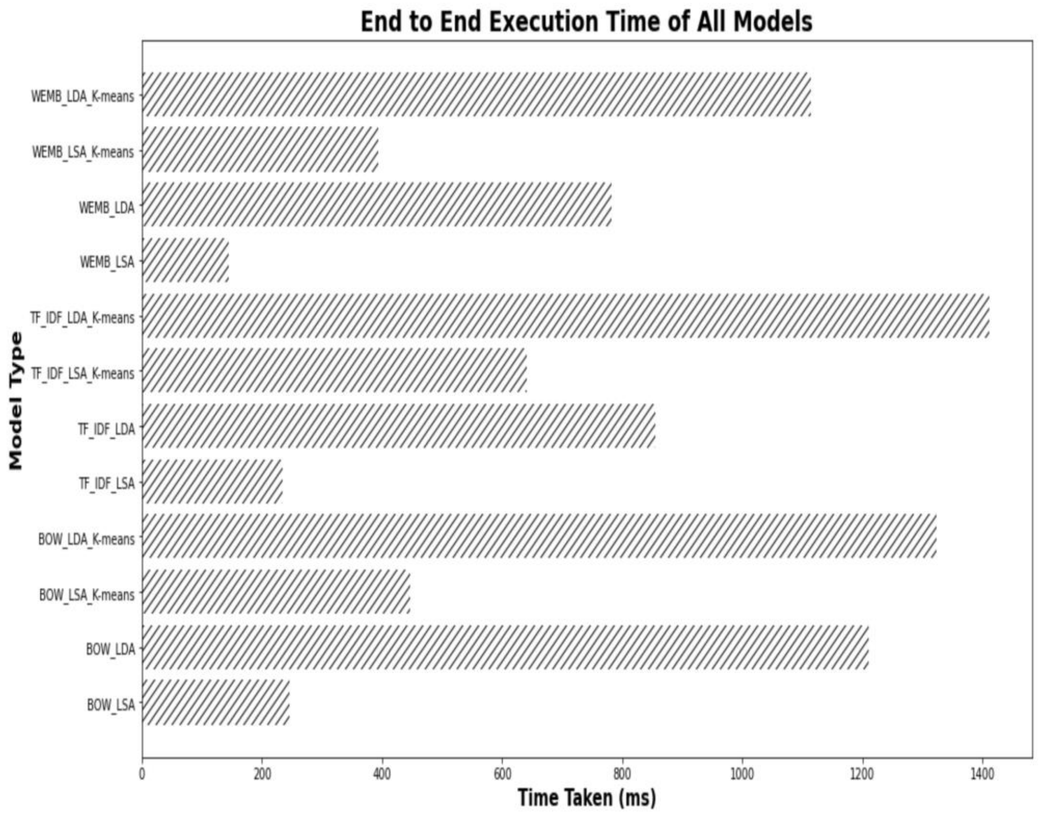


Figure 18. End to End execution time of all models on google cloud virtual machine.

Discussion

This subsection first answers the research questions identified during the preliminary research phase. Then this section will include further analysis of the results section.

a) *RQ-1:* What metrics exist to score semantic sentence similarities?

Equation 5 was used to generate scores for sentences in the corpus in the paper written by Padmakumar et al. (3). This equation used outputs of the LSA. In this equation, both matrices might have been squared to force all the values to be positive. Squares of Matrix U and S might have been multiplied to maximize the difference between high scores and low scores.

b) *RQ-2:* How do *n*-grams impact extractive summarization generation?

From figures 8-11, F-measure scores for BOW feature extraction schemes decreases as the *n*-gram range increases. If the BOW feature extraction scheme is to be used, then using 1-gram or 2-gram is probably a good idea. From figures 8-11, F-measure scores for TF-IDF fluctuates in a downward trend as *n*-gram range increases. If the TF-IDF feature extraction scheme is to be used, then using either 1-Gram or 3-Gram is probably a good idea. From figures 8-11, f-measure scores for word-embeddings feature extraction schemes decrease as the *n*-gram range increases.

c) *RQ-3:* How effective is the *word-embeddings* approach in generating features?

These two schemes are still highly used, however, can modern schemes like word-embeddings perform better than TF-IDF and BOW?

Word-embeddings work great when the machine learning models used are Neural Networks because no additional transformation is required to generate feature vectors. However, in this research, the outputs of the word-embeddings scheme were transformed into a sparse feature vector. Perhaps because of this sparsity, it can be inferred from figure 8-11, that BOW and TF-IDF scheme performed much better than word-embeddings scheme.

From Fig 8-11, it can be inferred that 1-gram summaries were the best performing summaries across all methods. 3-gram and 4-gram summaries also had satisfactory results across all methods. From figure 8, it can be inferred that when the summary was generated using LDA, TF-IDF worked as the best feature extraction scheme. From figure 9, when the summary was generated using LDA, the BOW feature extraction scheme worked as the best feature extraction scheme. From figure 10, when the summary was generated using LSA and *k*-means, TF-IDF worked as the best feature extraction scheme. Overall, figure 12 shows the comparison of the highest-scoring summaries. It can be inferred from figure 12 that, the method that had the highest rouge score was when the TF-IDF feature extraction scheme was used with LSA.

Computer analysis allowed us to rapidly visualize summary performance, however, these summaries are eventually read by human beings. So, a human-level assessment of these summaries would provide an in-depth analysis of the performance of these models. This required empirical analysis of these summaries. However, it was not feasible to have a human-level assessment of 60 different summaries. So, from computer level assessment, summaries generated using TF-IDF and LSA had the highest performance across 1-gram, 3-gram, and 4-gram features. So, a survey form containing these 3 summaries alongside the source text was sent out to be evaluated by people from various academic disciplines and professional backgrounds.

This analysis was done in form of a survey form where participants first read the source Wikipedia article about Hurricane Irene. Then the participants were presented with three selected summaries. The participants then evaluated those summaries based on the Likert Scale as shown in Fig 17.

Figure 13 shows the survey results for the summary generated using 1-gram TF-IDF as the feature generation scheme and LSA as the ML model. 78% of participants agreed this summary had an appropriate length for a user to understand the source text. There were mixed responses to whether the summary contained redundant information. 67% of participants agreed this summary was coherent. 56% of participants agreed that this summary illustrated the main ideas from the source text. In addition to that, 34% of respondents strongly agreed that this summary illustrated the main ideas from the source text. Finally, 89% of participants agreed that reading this summary once, conveyed the main idea of the source text.

Figure 14 shows the survey results for the summary generated using 3-gram TF-IDF as a feature generation scheme and LSA as the ML model. 56% of participants agreed this summary had an appropriate length for a user to understand the source text. There were mixed responses to whether the summary contained redundant information. However, the majority (56 %) of participants disagree with this summary not containing redundant information. 56% of participants agreed this summary was coherent. 67% of participants agreed that this summary illustrated the main ideas from the source text. Finally, 67% of participants agreed that reading this summary once, conveyed the main idea of the source text.

Figure 15 shows the survey results for the summary generated using 4-gram TF-IDF as feature generation scheme and LSA as ML model. 78% of participants agreed this summary had an appropriate length for a user to understand the source text. A majority (56%) of participants disagreed with this summary not containing redundant information. Mixed responses regarding whether this summary is coherent or not. 45% of participants agreed that this summary illustrated the main ideas from the source text. Finally, 45% of participants agreed while the other 45% of participants disagreed that reading this summary once, conveyed the main idea of the source text.

The code implementation and execution for this research was done on a google cloud virtual machine with CPU model: Intel(R) Xeon(R), CPU frequency: 2.30 GHz, number of CPU cores: 2, CPU family: Haswell, ram 12GB, and disk space: 25GB. Figure 18 shows the end-to-end execution time for all the models created or used in this research. This end-to-end execution time is a measurement of time for a model to extract features from an input corpus, train and fit a ML algorithm and finally generate a summary. Since each model produces different summaries for the 1-5 gram range, figure 18 shows the average end-to-end execution time taken across that range. From figure 18, it can be inferred that models, where *k*-means is used as the underlying ML algorithm, are the slowest. This is primarily because before using the feature vector to train the *k*-means model, the feature vector's dimensions are reduced using LSA or LDA, finally, the dimension reduced feature vector is used to train the *k*-means model and generate a summary. Head-to-head comparisons of models that used LSA or LDA are faster than other models that use *k*-means. In general, Figure 18 also shows models that use LDA are overall slower than models that use LSA. From the perspective of the feature extraction scheme, models that use the BOW scheme are the fastest, followed closely by models that use TF-IDF.

Computer level assessment based on ROUGE-N score shows the best performing models were the ones that used TF-IDF as feature extraction scheme and LSA as the underlying machine learning model. The empirical analysis results conform to the result generated by computer analysis which had the highest ROUGE-N score for this summary. The majority of participants disagreed with the statement that these summaries did not contain redundant information. This shows that the TF-IDF scheme does allow redundant information, which is accurate to the fact that the TF-IDF scheme generates huge sparse matrices, which might not always contain relevant information. All summaries seemed to be coherent, this establishes a fact for extractive summary, coherent information can be expected. All summaries convey the main ideas from the source text, this shows that this system captures the most relevant information from source texts.

Chapter 7: Conclusion and Future Tasks

Conclusion

In *extractive text summarization*, import sections of a source document are identified and concatenated to form a summary. In this project, an extractive text summarization system was built and analyzed. This system can summarize any Wikipedia article via web scraping. This project made use of the regular expression, word, and sentence tokenizers and lemmatizers to preprocess web scraped Wikipedia articles. Features were extracted from the preprocessed Wikipedia article using BOW, TF-IDF, and word-embeddings scheme for various co-occurrence windows (1,2,3,4,5-gram range). From the results and analysis section, it can be confirmed that either 3-gram or 4-gram should be used for the co-occurrence window. Even though there was no, head to comparisons of LSA and LDA. LSA when used with a mathematical model performed much better than LDA. Both computer evaluation and empirical analysis showed the TF-IDF scheme to be better than the BOW and word-embeddings scheme within the context of this research. From figure 18, even though models that used BOW were slightly faster than the ones that used TF-IDF, they are not significantly faster, that a speed versus accuracy tradeoff argument could be established. Overall, models that used TF-IDF as the feature extraction scheme produced the best summaries.

Future Tasks

The ROUGE-N recall scores were still below par for the summaries generated by the current system. These scores could be improved by using advanced preprocessing techniques such as part of speech tagging. LSA and LDA might be a good starting point for topic modeling, however Recurrent Neural Nets based models such as Transformers, and BERT could be used to generate better summaries.

Works Cited

Akter, Sumya, et al. "An Extractive Text Summarization Technique for Bengali Document(s) Using K-Means Clustering Algorithm." *2017 IEEE International Conference on Imaging, Vision Pattern Recognition (IcIVPR)*, 2017, pp. 1–6. *IEEE Xplore*, https://doi.org/10.1109/ICIVPR.2017.7890883.

Blei, David M., et al. "Latent Dirichlet Allocation." *The Journal of Machine Learning Research*, vol. 3, no. null, 2003, pp. 993–1022.

Chen, J., and H. Zhuge. "Extractive Text-Image Summarization Using Multi-Modal RNN." *2018 14th International Conference on Semantics, Knowledge and Grids (SKG)*, 2018, pp. 245–48. *IEEE Xplore*, https://doi.org/10.1109/SKG.2018.00033.

Dalal, V., and L. Malik. "A Survey of Extractive and Abstractive Text Summarization Techniques." *2013 6th International Conference on Emerging Trends in Engineering and Technology*, 2013, pp. 109–10. *IEEE Xplore*, https://doi.org/10.1109/ICETET.2013.31.

IBM Cloud Education. *What Is Machine Learning?* 15 July 2020, https://www.ibm.com/cloud/learn/machine-learning.

Jain, A., et al. "Extractive Text Summarization Using Word Vector Embedding." *2017 International Conference on Machine Learning and Data Science (MLDS)*, 2017, pp. 51–55. *IEEE Xplore*, https://doi.org/10.1109/MLDS.2017.12.

Kumar, Manoj. *Emerging Trends in Big Data, IoT and Cyber Security*. p. 258.

Landauer, Thomas K., et al. "An Introduction to Latent Semantic Analysis." *Discourse Processes*, vol. 25, no. 2–3, 2–3, Jan. 1998, pp. 259–84. *DOI.org (Crossref)*, https://doi.org/10.1080/01638539809545028.

Lin, Chin-Yew. *ROUGE: A Package for Automatic Evaluation of Summaries*. July 2004, p. 8.

Moratanch, N., and S. Chitrakala. “A Survey on Extractive Text Summarization.” *2017 International Conference on Computer, Communication and Signal Processing (ICCCSP)*, 2017, pp. 1–6. *IEEE Xplore*, https://doi.org/10.1109/ICCCSP.2017.7944061.

Padmakumar, Aishwarya, and Dhivya Eswaran. *Extractive Text Summarization Using Latent Semantic Analysis*. 2014, p. 10.

Padmakumar, Aishwarya, and Akanksha Saran. *Unsupervised Text Summarization Using Sentence Embeddings*. p. 9.

Widjanarko, Agus, et al. “Multi Document Summarization for the Indonesian Language Based on Latent Dirichlet Allocation and Significance Sentence.” *2018 International Conference on Information and Communications Technology (ICOIACT)*, 2018, pp. 520–24. *IEEE Xplore*, https://doi.org/10.1109/ICOIACT.2018.8350668.

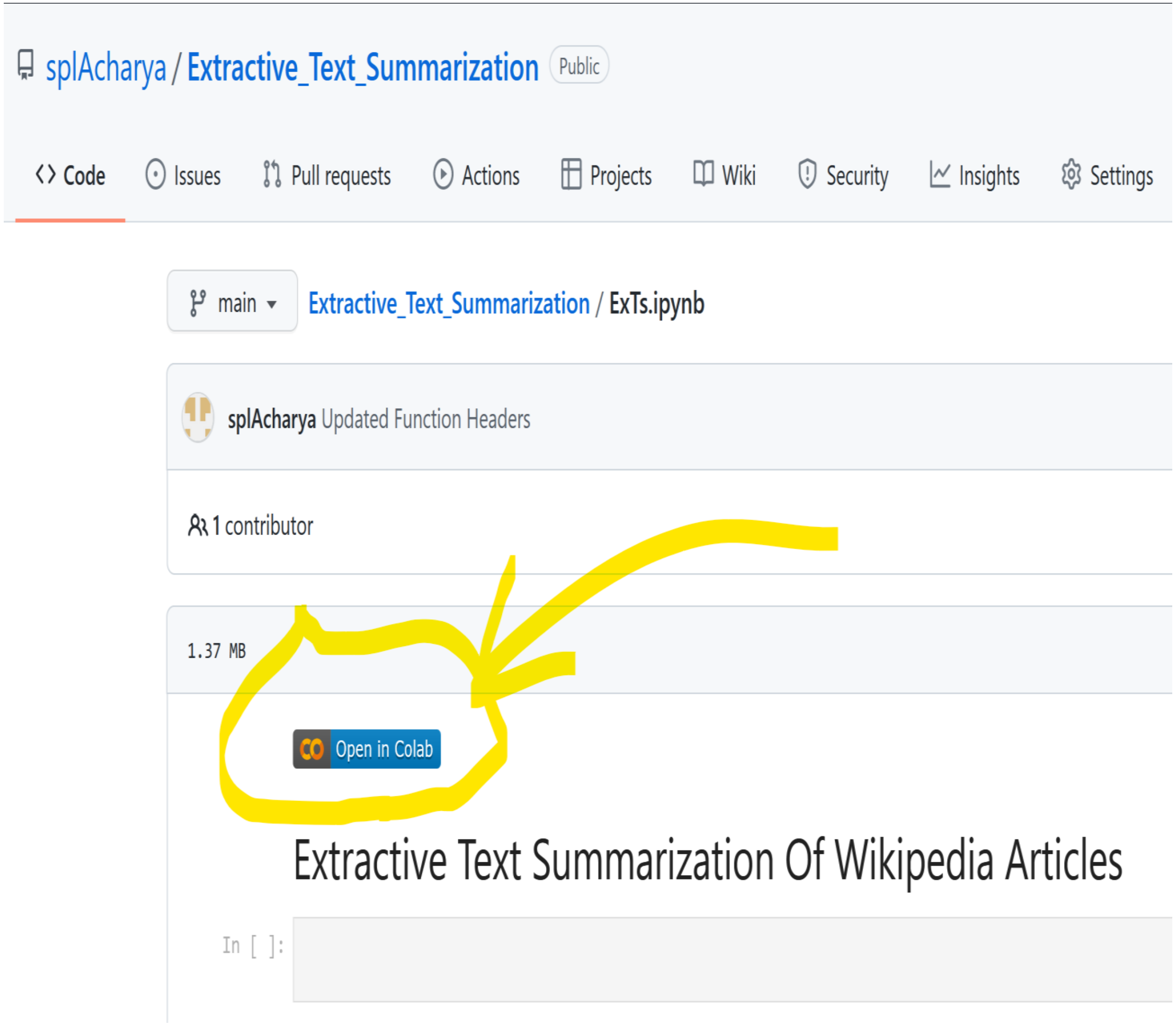
Appendix A: Recreating Results

Overview

This section shows how to recreate the summaries generated in this research. The first step would be to open the following GitHub link:

https://github.com/splAcharya/Extractive_Text_Summarization/blob/main/ExTs.ipynb

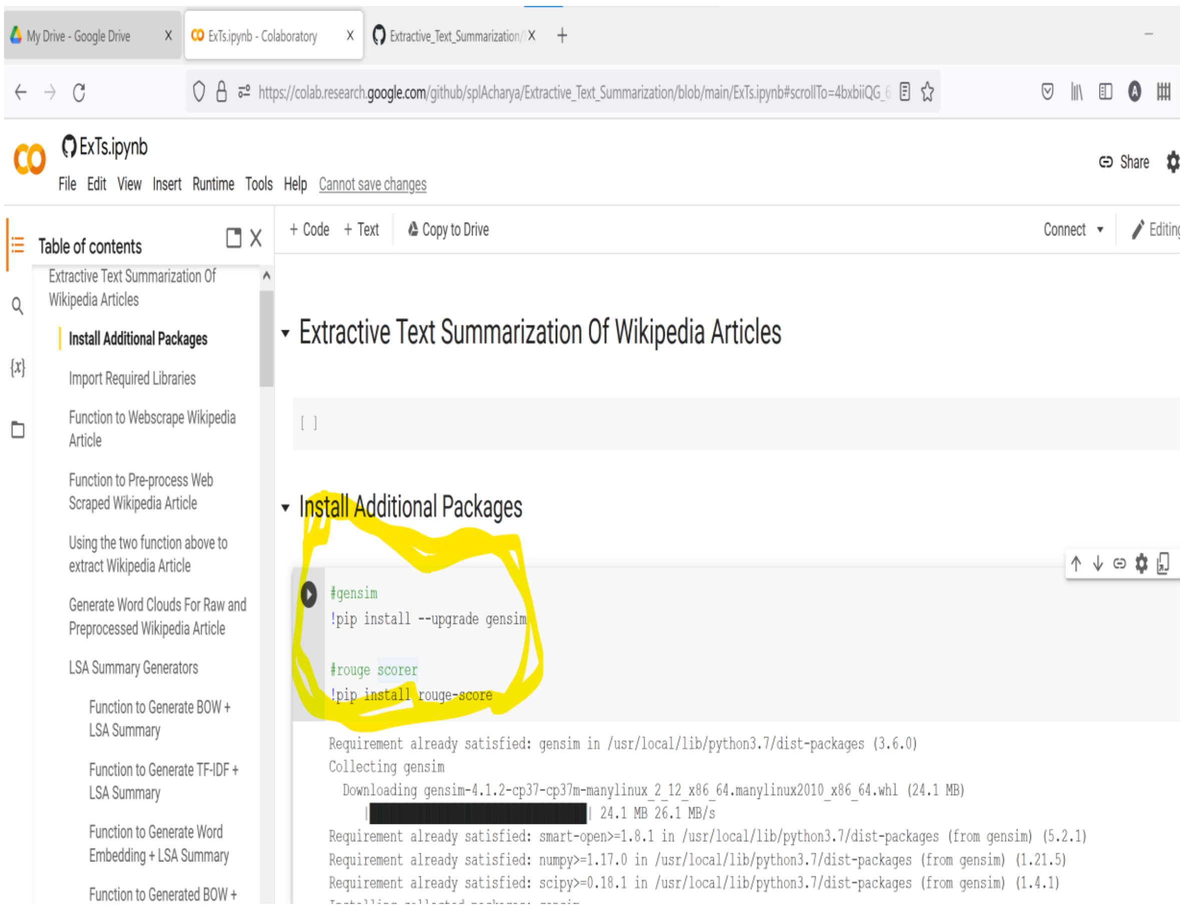
When the link above is clicked, the web browser should open the page as shown in the image below. Then click on the *open in colab* button highlighted by yellow in the image below.


splAcharya / Extractive_Text_Summarization Public
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
main
Extractive_Text_Summarization / ExTs.ipynb
splAcharya Updated Function Headers
1 contributor
1.37 MB
Open in Colab
Extractive Text Summarization Of Wikipedia Articles
In [ ]:

Once the *open in colab* button is clicked, you see this project open in google collaboratory. To run this project in google colab, it will be necessary to sign in with a google email. Once the sign in is completed, your web browser should show you google colab page as shown in the following section.

Setting up environment

Before trying to run the whole project, additional libraries would need to be installed, some libraries/packages are provided by default in google colab. Others have been coded into the project so no special care would be needed. The highlighted portion in the image below shows the commands that install additional packages.


My Drive - Google Drive
ExTs.ipynb - Colaboratory
Extractive_Text_Summarization/
https://colab.research.google.com/github/splAcharya/Extractive_Text_Summarization/blob/main/ExTs.ipynb#scrollTo=4bxbiiQG_
ExTs.ipynb
File Edit View Insert Runtime Tools Help Cannot save changes
Share
Table of contents
Extractive Text Summarization Of Wikipedia Articles
Install Additional Packages
Import Required Libraries
Function to Webscrape Wikipedia Article
Function to Pre-process Web Scraped Wikipedia Article
Using the two function above to extract Wikipedia Article
Generate Word Clouds For Raw and Preprocessed Wikipedia Article
LSA Summary Generators
Function to Generate BOW + LSA Summary
Function to Generate TF-IDF + LSA Summary
Function to Generate Word Embedding + LSA Summary
Function to Generated BOW +
+ Code + Text
Copy to Drive
Connect
Editing

Extractive Text Summarization Of Wikipedia Articles

Install Additional Packages

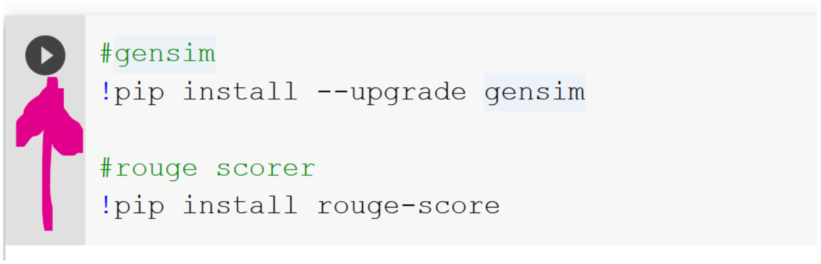
```
#gensim
!pip install --upgrade gensim

#rouge scorer
!pip install rouge-score
```

```
Requirement already satisfied: gensim in /usr/local/lib/python3.7/dist-packages (3.6.0)
Collecting gensim
  Downloading gensim-4.1.2-cp37-cp37m-manylinux_2_12_x86_64.manylinux2010_x86_64.whl (24.1 MB)
     |████████████████████████████████| 24.1 MB 26.1 MB/s
Requirement already satisfied: smart-open>=1.8.1 in /usr/local/lib/python3.7/dist-packages (from gensim) (5.2.1)
Requirement already satisfied: numpy>=1.17.0 in /usr/local/lib/python3.7/dist-packages (from gensim) (1.21.5)
Requirement already satisfied: scipy>=0.18.1 in /usr/local/lib/python3.7/dist-packages (from gensim) (1.4.1)
```

Running the code

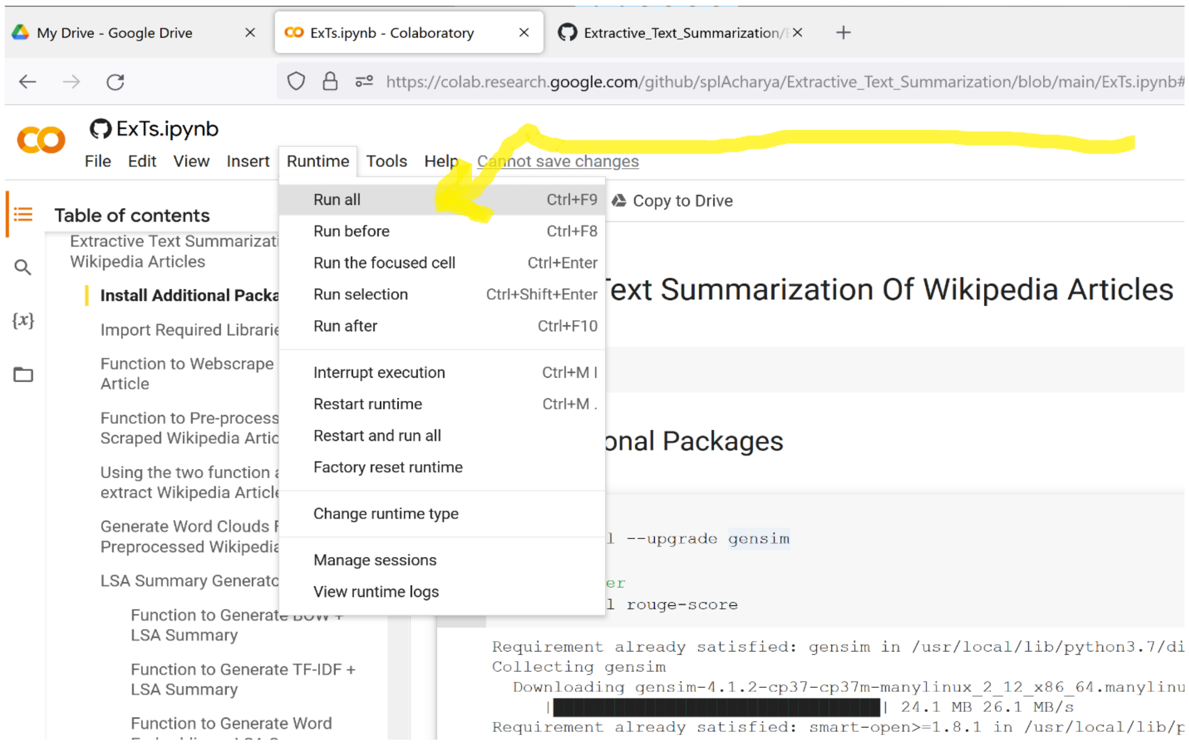
To run each *cell* individually in google colab, one can either hover over that cell and press the *play* button as shown in the image below. Or press *Shift + Enter* key together.



```
#gensim
!pip install --upgrade gensim

#rouge scorer
!pip install rouge-score
```

The above image shows how to run individual cells, however, if it is desired to run the entire project at once to see outputs. In the menu bar, then click *runtime* and click *run all* (see image below). This should automatically run the entire project at once. This step should take 4-5 minutes to complete. After 4-5 minutes, one can simply scroll down the page to see details of this project.



```
l --upgrade gensim

er
l rouge-score
```

```
Requirement already satisfied: gensim in /usr/local/lib/python3.7/di
Collecting gensim
  Downloading gensim-4.1.2-cp37-cp37m-manylinux_2_12_x86_64.manylinu
     |████████████████████████████████| 24.1 MB 26.1 MB/s
Requirement already satisfied: smart-open>=1.8.1 in /usr/local/lib/p
```