

**TSDA: Topic Stability Driven Approach with Data
Flow Analysis and Hyperparameter Optimization
For Enhanced Topic Sentiment Summarization**
A PROJECT REPORT

Submitted by

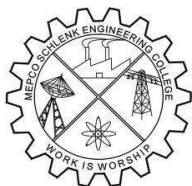
T.AJAY KUMAR	(Reg. No. 202006005)
P.RAJA ATHIBAN	(Reg. No. 202006036)
P.K.VASANTH RAMM	(Reg. No. 202006254)

*in partial fulfillment for the award of the degree
of*

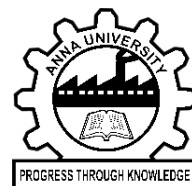
BACHELOR OF TECHNOLOGY

in

INFORMATION TECHNOLOGY



DEPARTMENT OF INFORMATION TECHNOLOGY
MEPCO SCHLENK ENGINEERING COLLEGE, SIVAKASI
(An Autonomous Institution affiliated to Anna University, Chennai)



April 2024

BONAFIDE CERTIFICATE

Certified that this project report titled **TSDA: Topic Stability Driven Approach with Data Flow Analysis and Hyperparameter Optimization For Enhanced Topic Sentiment Summarization** is the bonafide work of **T.Ajay Kumar (Regno: 202006005)**, **P.Raja Athiban (Regno: 202006036)**, **P.K.Vasanth Ramm (Regno: 202906254)** who carried out the research under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

Mrs.S.AISHWARYA, M.E.,
Internal Guide
Assistant Professor,
Department of Information Technology,
Mepco Schlenk Engineering College,
Sivakasi-626005.
Virudhunagar Dt.
Tamilnadu.

Dr.T.REVATHI, M.E., Ph.D.,
Head of the Department
Senior Professor,
Department of Information Technology,
Mepco Schlenk Engineering College,
Sivakasi-626005.
Virudhunagar Dt.
Tamilnadu.

Submitted for Viva-Voce Examination held at **MEPCO SCHLENK ENGINEERING COLLEGE, SIVAKASI (AUTONOMOUS)** on

Internal Examiner

External Examiner

ABSTRACT

ABSTRACT

Our study introduces a new method called Topic Stability Driven Approach with Data Flow Analysis and Hyperparameter Optimization. This method aims to make the topics we find in documents more stable and reliable. We use two types of Topic models: LDA and LSA. To improve these models, we apply techniques like Grid Search and Genetic Algorithm to optimize their settings. To ensure the topics we find are stable, we evaluate them using coherence score metrics for each document. This helps us make sure the topics make sense and stay consistent across different analyses. We also work on sentiment analysis, which is understanding the emotions expressed in text. We also analyze how sentiments change across different topics in the text, which helps us understand people's feelings better. Lastly, we fine-tune our methods using hyperparameter optimization. This helps us make sure our techniques work well for different kinds of text and topics. Our experiments show that our approach, TSDA works well across different datasets and topics. By using this method, we can better understand how people feel about specific subjects, which is important for many applications. Overall, our method provides a promising way to improve topic sentiment summarization, making it more useful and reliable in various real-world situations.

ACKNOWLEDGEMENT

ACKNOWLEDGEMENT

Apart from our efforts, the success of our project depends largely on the encouragement of many others. We take this opportunity to express our gratitude to the people who have been instrumental in the successful completion of our project. We would like to express our immense pleasure to thank our college management for giving required amenities regarding our project.

We would like to convey our sincere thanks to our respected Principal, **Dr.S.Arivazhagan, M.E.,Ph.D.**, Mepco Schlenk Engineering College, for providing us with facilities to complete our project.

We extend our profound gratitude and heartfelt thanks to **Dr.T.Revathi, M.E.,Ph.D.**, Senior Professor and Head of the Department of Information Technology for providing us constant encouragement.

We are bound to thank our project coordinator **Dr.Rajesh M.E.,MBA.,Ph.D.**, Professor of Information Technology. We sincerely thank our project guide **Mrs. S.Aishwarya M.E.**, Assistant Professor, Department of Information Technology, for her inspiring guidance and valuable suggestions to complete our project successfully.

The guidance and support received from all the staff members and lab technicians of our department who contributed to our project, was vital for the success of the project. We are grateful for their constant support and help.

We would like to thank our parents and friends for their help and support in our project.

TABLE OF CONTENTS

TABLE OF CONTENTS

	CONTENT	PAGE NO
LIST OF TABLES		x
LIST OF FIGURES		xii
LIST OF SYMBOLS		xiv
LIST OF ABBREVIATIONS		xvi
 CHAPTER 1	 INTRODUCTION	 1
1.1	SENTIMENT ANALYSIS	2
1.2	TSDA FRAMEWORK	2
1.3	TOPIC MODELING	3
1.4	HYPERPARAMETER OPTIMIZATION TECHNIQUES	3
1.5	DATA FLOW ANALYSIS	4
CHAPTER 2	LITERATURE STUDY	5
2.1	OVERVIEW	6
2.2	UNSUPERVISED CONDENSED ABSTRACTION	6
2.3	PARAMETER TUNING	7
2.4	SENTIMENT ANALYSIS BY POS	7
2.5	THEME EXTRACTION	8
2.6	OPINION MINING	9
2.7	DATA FLOW ANALYSIS	10
CHAPTER 3	SOFTWARE REQUIREMENTS SPECIFICATION	12
3.1	SCOPE	13
3.2	METHODOLOGIES	13
3.3	SYSTEM REQUIREMENTS	14
	3.3.1 HARDWARE INTERFACES	14
	3.3.2 SOFTWARE INTERFACES	14

3.4	FUNCTIONAL REQUIREMENTS	15
3.5	NON-FUNCTIONAL REQUIREMENTS	17
CHAPTER 4	SYSTEM DESIGN	19
4.1	OVERVIEW	20
4.2	OVERALL ARCHITECTURE	20
4.3	MODULES	20
4.3.1	DATA PREPROCESSING	21
4.3.2	DOCUMENT TERM MATRIX	22
4.3.3	TOPIC MODELING	23
4.3.4	HYPERPARAMETER OPTIMIZATION	24
4.3.5	SENTIMENT CALCULATION	25
4.3.6	DATA FLOW ANALYSIS	26
4.3.7	SENTIMENT CLASSIFICATION	27
CHAPTER 5	IMPLEMENTAION METHODOLOGY	29
5.1	OVERVIEW	30
5.2	ESSENTIAL LIBRARIES	30
5.2.1	GENSIM	30
5.2.2	TEXTBLOB	30
5.2.3	TRUNCATED SVD	31
5.2.4	WORDNET LEMMATIZER	31
5.2.5	SCIKIT LEARN	31
5.2.6	SWIFTER	32
5.2.7	AST (ABSTRACT SYNTAX TREE)	32
5.2.8	NLTK (NATURAL LANGUAGE TOOLKIT)	32
5.2.9	BEAUTIFUL SOUP	32
5.2.10	SEABORN	32
5.2.11	MATPLOTLIB	33
5.2.12	NUMPY	33
5.2.13	WORDCLOUD	33
5.2.14	RE (REGULAR EXPRESSION)	33
5.3	FUNCTIONS USED FOR IMPLEMENTATION	34
5.3.1	PREPROCESSING	34
5.3.2	SENTIMENT CLASSIFICATION	34
5.3.3	GENETIC ALGORITHM	35
5.3.4	GRID SEARCH	35
5.3.5	REACHING DEFINITION	36
5.3.6	SUMMARY	36
5.4	FRAMEWORKS	37

CHAPTER 6	PERFORMANCE METRICS	38
6.1	OVERVIEW	39
6.2	DIRECT CONFUSION MATRIX	39
6.3	RELEVANCY AND FLUENCY	40
6.4	ROUGE SCORE	40
6.5	COHERENCE SCORE	41
CHAPTER 7	RESULTS AND DISCUSSION	42
7.1	OVERVIEW	43
7.2	DATASET	43
	7.2.1 STATIONARY DATASET	43
7.3	SCREENSHOTS	44
CHAPTER 8	CONCLUSION AND FUTURE WORK	52
8.1	CONCLUSION	53
8.2	FUTURE WORK	51
CHAPTER 9	APPENDIX	52
9.1	CODING	53
CHAPTER 10	REFERENCES	63
10.1	REFERENCES	64
CHAPTER 11	ANNEXURE	66
11.1	JOURNAL SUBMISSION PROOF	67
11.2	PAPER PLAGIARISM PROOF	68
11.3	REPORT PLAGIARISM	69

LIST OF TABLES

LIST OF TABLES

TABLE NO	TABLE NAME	PAGE NO
7.2.1.1	Data Samples used for each frameworks	43
7.3.15	Best Parameters for LDA Model from Grid Search	51
7.3.16	Best Configurations of Parameters for No of Topics from Genetic Algorithm for LDA Model	51

LIST OF FIGURES

LIST OF FIGURES

FIGURE NO.	TOPIC	PAGE NO.
3.2.1	Activity Diagram for our project	13
4.2.1	Overall Workflow of TSDA Framework	20
4.3.1.1	Steps involved in Data Preprocessing	21
4.3.3.1	Word Cloud for Topics and Topic words	23
4.3.3.2	Topic modeling encompasses both LDA & LSA	24
4.3.4.1	Finding Best Parameters for LDA Model	24
4.3.4.2	Finding Best Parameters for LSA Model	25
7.3.1	LDA Nexus – Topic Stability of a document	44
7.3.2	LSA Nexus – Topic Stability of a document	44
7.3.3	GenAlgo LDA – Average coherence score vs GA Configurations	45
7.3.4	LDA Topics vs Mean Accuracy	45
7.3.5	TopicBurst LDA – LDA Topics vs Accuracy using Direct CM Analysis	46
7.3.6	TopicBurst LSA – LSA Topics vs Accuracy using Direct CM Analysis	46
7.3.7	TopicBurst LDA – LDA Topics vs Fluency and Relevance Score	47
7.3.8	TopicBurst LSA – LSA Topics vs Fluency and Relevance Score	47
7.3.9	GenAlgo LDA – Configurations vs Fluency and Relevance Score	48
7.3.10	ML and Cross Validation vs Frameworks with and without Ensemble of models with best setting	48
7.3.11	Accuracy vs Frameworks with best settings	49
7.3.12	Relevance Score vs Frameworks with best setting	49
7.3.13	Fluency Scores vs Frameworks with best settings	50
7.3.14	Rouge 1,2, L Scores vs Frameworks	50

LIST OF SYMBOLS

LIST OF SYMBOLS

NOTATION	MEANING
X	Dataset
w_i	Weights
P	Precision
F	F1-Score
R	Recall

LIST OF ABBREVIATION

LIST OF ABBREVIATION

S.NO	ACRONYMS	ABBREVIATIONS
1	VADER	Valence Aware Dictionary sEntiment Reasoner
2	FP	False Positive
3	TP	True Positive
4	NLP	Natural Language Processing
5	FN	False Negative
6	FP	False Positive
7	TSDA	Topic Stability Driven Approach

INTRODUCTION

CHAPTER 1

INTRODUCTION

1.1 SENTIMENT ANALYSIS

In today's modern landscape flooded with textual data, the understanding of sentiments and opinions expressed in written content has become increasingly essential. From social media updates to news articles, texts serve as data collections of valuable insights into people's feelings and attitudes towards various subjects. Sentiment analysis, a process aimed at extracting and understanding emotions from text, assumes a crucial role in understanding the underlying sentiments hidden within large amounts of textual information. One crucial application of sentiment analysis lies in topic sentiment summarization, which involves extracting the sentiments expressed about specific subjects within a corpus of text.

1.2 TSDA FRAMEWORK

Our paper presents a novel technique termed as Topic Stability Driven Approach with Data Flow Analysis and Hyperparameter Optimization to help summarize the feelings about different topics in documents. It combines ideas from stable methods, of understanding how data moves, and techniques to find the best settings automatically. This helps us get a deeper understanding of how people feel about specific topics in what they write online. To achieve this, we've used two types of models: LDA and LSA. These models help us identify the main themes in a document. We have also applied techniques like Grid Search and Genetic Algorithm to make these models work even better. These techniques help us find the best settings for our models automatically. The TSDA framework represents a significant advancement in sentiment analysis methodology, offering a comprehensive solution for summarizing the sentiments associated with different topics within textual documents.

1.3 TOPIC MODELING

Topic modeling serves as a method for uncovering the predominant themes or subjects within a compilation of documents. In our study, we use two common techniques for topic extraction: LDA and LSA.

LDA serves as statistical framework employed to reveal hidden topics within a corpus of documents. It functions based on the premise that separate documents comprise a blend of dissimilar topics, with every word in document linked to one of these topics. LDA works by iteratively assigning words to topics and adjusting the topic assignments to enhance the potential of observing the actual spread of words in the documents. Through this iterative process, LDA uncovers the underlying topics and their associated word distributions. These topics are represented as probability distributions over words, indicating the likelihood of a word appearing in a document given a particular topic. Overall, LDA provides a powerful framework for automatically discovering the latent themes or topics present in large text corpora.

LSA is a method used for understanding the relationships between words and documents in an extensive corpus of textual data. It works by analyzing the frequency of word occurrences across documents and representing them in a high-dimensional matrix. LSA utilizes Singular-Value-Decomposition (SVD) to condense the feature space of the matrix while preserving vital semantic correlations. This reduction process helps to uncover latent, or hidden, semantic structures within the text data. By identifying patterns of word co-occurrence, LSA can capture the underlying meaning or context of words. This means that even if words don't appear together frequently, LSA can still recognize similarities in their usage and group them accordingly. In essence, LSA enables us to understand the semantic associations between words and documents.

1.4 HYPERPARAMETER OPTIMIZATION TECHNIQUES

Hyperparameter optimization involves adjusting the parameters of a model to enhance its performance to the highest possible level. In our study, we focus on improving the performance of our topic modeling method by adjusting parameters such as alpha, beta, decay, random state, min probability, which we call hyperparameters.

Think of hyperparameters as dials on a machine that controls how it operates. Two techniques we use for this optimization are Grid Search and Genetic Algorithm. Grid Search is a systematic way of trying out different combinations of hyperparameters to see which ones work best. We create a grid of possible hyperparameter values and test each combination to see which gives us better negative log-likelihood score. On the other hand, GA mirrors the process of Darwinian selection to find the optimal hyperparameters. It starts with a population of potential solutions (sets of hyperparameters) and evaluates their performance. Then, it selects the best-performing solutions and combines them to create new ones. This process continues over several generations until it finds the best combination of hyperparameters, like evolution refining the traits of a species over time. By using these techniques, we ensure that our topic modeling method works effectively across different types of text and topics. This optimization process is crucial for making our method more accurate and reliable, ultimately leading to better understanding of people's sentiments and more useful insights in real-world applications.

1.5 DATA FLOW ANALYSIS

Dataflow analysis is like following a path of information as it moves through a program. Imagine each piece of information as a little traveler going through different parts of a journey. We're interested in tracking how these travelers move, where they go, and how they influence each other. By doing this, we can understand how data is used and transformed in a program. It helps us spot potential errors, optimize performance, and improve security. Essentially, it's about understanding the flow of data within a program to make it work better and more efficiently.

Data flow analysis in summarization entails extracting pertinent information from a text to create a concise summary, relying on Sentence-topic and Topic-word distribution. Sentence- topic distribution involves mapping sentences to the topics they address, aiding in identifying key themes. Topic-word distribution complements this by analyzing word distribution within topics, revealing crucial terms and concepts. The process involves text preprocessing to remove irrelevant elements, followed by topic modeling using techniques like LDA or NMF to uncover underlying topics and establish distributions.

CHAPTER 2

LITERATURE STUDY

2.1 OVERVIEW

The Topic Stability Driven Approach (TSDA), which integrates Data Flow Analysis and Hyperparameter Optimization to enhance topic sentiment summarization. Employing LDA and LSA models, the method prioritizes stability and reliability of topics, leveraging techniques like Grid Search and Genetic Algorithm for optimization. Coherence score metrics ensure consistent and meaningful topics across analyses. Additionally, sentiment analysis is utilized to discern emotions within text, observing sentiment changes across topics. Fine-tuning through hyperparameter optimization ensures applicability across diverse texts. Experimental results demonstrate TSDA's efficacy across datasets, enhancing understanding of sentiment towards specific subjects for various applications.

2.2 UNSUPERVISED CONDENSED ABSTRACTION

Text abridgment is a crucial task in NLP, aiming to distill any prolonged text into concise synopses while retaining essential information. However, many deep learning models for this task require abundant labeled data, which is often lacking, especially for lesser-known languages. Additionally, training such models demands substantial computational resources due to their intricate architectures. In response to these challenges, LFIP-SUM emerges as an innovative unsupervised summarization model. Unlike its supervised counterparts, LFIP-SUM operates without the need for labeled data during training. Instead, it leverages pre-trained sentence embeddings and integer programming techniques. By employing principal component analysis (PCA), the model autonomously identifies and selects significant sentences to include in the summary. Despite its lack of parameter learning, experiments demonstrate that LFIP-SUM achieves comparable performance to deep learning-based summarization models. This approach represents a promising solution, particularly in resource-constrained environments where labeled data and computational power are scarce. LFIP-SUM's ability to generate accurate summaries without the need for extensive training data or high computational costs makes it a valuable tool for various applications, including those requiring summarization in lesser-known languages or under resource limitations.

2.3 PARAMETER TUNING

Machine learning models have become indispensable in various domains, prompting researchers to continuously seek methods to enhance their accuracy. In this pursuit, the exploration of different approaches for adjusting hyperparameters is imperative. Our study delves into comparing two prominent techniques: Grid Search and Genetic Algorithm (GA). Grid Search systematically explores all possible combinations of hyperparameters, ensuring thorough coverage but often at the cost of extensive computational time, particularly with complex models. On the other hand, Genetic Algorithms emulate the principles of natural evolution, iteratively refining hyperparameters over successive generations. We evaluate the efficacy of these methods in the context of Arabic sentiment classification, a task known for its intricacies owing to the intricacy of the Arabic language. By undertaking this analysis, we shed light on the respective strengths and limitations of Grid Search and Genetic Algorithm in fine-tuning machine learning models for sentiment analysis. Through our investigation, we aim to provide valuable insights into the optimization of machine learning models in practical scenarios. By understanding the performance characteristics of these tuning techniques, practitioners are capable of making knowledgeable choices regarding the selection and application of hyperparameter optimization methods, ultimately optimizing the potency of sentiment analysis tasks in Arabic and potentially other complex languages.

2.4 SENTIMENT ANALYSIS BY POS

Sentiment Analysis using grammatical category (POS) tagging is a sophisticated method employed to decipher the emotional tone embedded within textual data. This process entails dissecting sentences into their constituent words and discerning their grammatical functions, including nouns, verbs, adjectives, and more. Through this meticulous examination of the sentence structure, the sentiment conveyed can be inferred. In essence, words like "happy," "joyful," or "satisfied" typically evoke positive sentiments, contrasting with terms such as "sad," "angry," or "disappointed," which commonly convey negative emotions. By scrutinizing the occurrence and context of these sentiment-laden words within a sentence, Sentiment Analysis facilitates astute evaluations concerning whether the prevailing sentiment leans towards positivity, negativity, or neutrality. This analytical approach not only captures the surface-level sentiment but also delves into the underlying nuances, thereby furnishing a richer understanding of the emotional content encapsulated in textual data.

Such insights gleaned from Sentiment Analysis find diverse applications, spanning from deciphering customer feedback sentiments to monitoring social media conversations, thereby empowering businesses, and organizations to make data-driven decisions and cultivate meaningful interactions with their audience.

2.5 THEME EXTRACTION

Latent Dirichlet Allocation (LDA) serves as a powerful method for unraveling the underlying structure within large textual datasets, such as articles or books. Its mechanism involves the conceptualization of each article or text as a combination of diverse topics, wherein each Topic represents a fusion of various words. Through this approach, LDA enables the identification of latent themes or topics present in the corpus and quantifies their significance within each document. By uncovering these hidden mixes of topics and words, LDA facilitates a deeper understanding of the content's thematic composition. This capability proves invaluable for a multitude of tasks, ranging from organizing articles to gaining insights into their subject matter. Moreover, LDA's ability to reveal latent themes amidst a vast collection of writings extends its utility to tasks like content recommendation systems, where understanding the underlying themes aids in suggesting related or similar articles to users. In essence, LDA provides a means to distill complex textual data into interpretable topic distributions, thereby enabling enhanced comprehension, organization, and utilization of large-scale text corpora for diverse applications in research, information retrieval, and content management.

Nonnegative Matrix Factorization (NMF) offers a versatile approach to text analysis by extracting latent patterns in data without imposing strong assumptions. A novel advancement in this realm, Deep NMF (DNMF), introduces a sophisticated framework by integrating deep learning techniques. DNMF leverages deep learning methodologies to uncover intricate structures within documents, enabling a more nuanced understanding of textual content. By employing deep learning models, DNMF autonomously learns hidden representations embedded within the text, capturing complex relationships and semantics. Subsequently, this acquired knowledge is harnessed to identify crucial words associated with specific topics, facilitating more precise topic modeling. The effectiveness of DNMF is demonstrated through

empirical evaluations across diverse text corpora, showcasing its superiority over traditional methods. This approach not only enhances the interpretability and accuracy of topic modeling but also reduces the reliance on manual feature engineering and model assumptions. DNMF's ability to adapt to different types of text data underscores its versatility and applicability across various domains. Through its innovative integration of deep learning and NMF, DNMF represents a significant advancement in text analysis, promising more insightful and robust topic modeling capabilities for researchers and practitioners alike.

Topical n-grams represent a sophisticated theme extraction method that extends beyond the conventional Bo-W approach. Unlike Bo-W, which treats each word in isolation, topical n-grams delve deeper into the text by considering not only individual words but also phrases of varying lengths (n-grams), acknowledging the significance of word order in conveying meaning. This technique involves sampling topics and their associated word statuses, which can be unigrams (single words) or bigrams (two-word phrases), in sequential fashion. By doing so, the model can discern meaningful phrases within topics, thus yielding more interpretable and contextually rich thematic representations. The incorporation of n-grams enables the model to capture nuanced linguistic structures and semantic relationships present in the text, thereby enhancing its ability to discern fine-grained themes. Consequently, this approach facilitates improved performance across various natural language processing tasks, particularly in areas such as information retrieval, where the extraction of relevant and semantically coherent topics is paramount. Through the utilization of topical n-grams, analysts can gain deeper insights into the underlying structure and content of textual data, enabling more effective knowledge extraction and decision-making processes.

2.6 OPINION MINING

Sentiment Analysis alternatively termed as Opinion Mining. Its emphasis lies in discerning emotional nuances within text, notably on platforms such as Twitter, where opinions are plentiful yet frequently disorganized. This survey explores techniques for analyzing Twitter data to determine whether opinions are positive, negative, or neutral, using methods like ML models and Word-Inventory based approaches, alongside scoring methods and various algorithms such as N-Bayes, and Support Vector Machine.

It can indeed be applied in education to evaluate students' learning experiences, but current algorithms may not fully replace human raters. In a study using various machine learning algorithms to analyze students' sentiments about learning experiences, results showed high accuracy in identifying positive and negative sentiments but struggled with neutral sentiments. However, an algorithm using word-sentiment associations achieved decent accuracy without needing extensive pretraining, indicating potential for improvement with more educational datasets. It is a field within natural language processing, focuses on understanding human emotions and opinions. This study examines how large language models like ChatGPT perform in various sentiment analysis tasks, finding that while they excel in simpler tasks, they struggle with more complex analyses. Additionally, the study proposes a new benchmark for evaluating sentiment analysis models.

2.7 DATAFLOW ANALYSIS

In modern compilers, the Static Single Assignment (SSA) form helps analyze and optimize programs. Converting programs into SSA involves placing ϕ -functions, which can be complex. Existing methods, like dominance frontiers (DF), assume all variables start at the program's beginning, which isn't always true for local variables.

A new ϕ -placement algorithm based on data flow analysis (DFA) is introduced in this paper to address this issue. The correctness and complexity of this approach are demonstrated through theorems and proofs. The introduction of the new ϕ -placement algorithm based on data flow analysis (DFA) marks a significant advancement in compiler technology. By addressing the limitations of existing methods like dominance frontiers (DF), which assume uniform variable initialization, this algorithm promises greater accuracy in program analysis. Through rigorous testing in the Clang/LLVM compiler framework, the approach showcased notable outcomes, significantly reducing unnecessary ϕ functions and enhancing precision. Despite the increased computational demands, the method's efficiency in analyzing a substantial proportion of procedures. One fundamental formula used in data flow analysis is the iterative solution for computing the 'in' and 'out' sets for each program point in a control flow graph. These sets represent the information flowing into and out of each program point, respectively. The iterative equations for computing the 'in' and 'out' sets are as follows:

1. In sets: $\text{In}[S] = \text{Union of } \{\text{Out}[P] \mid P \text{ Belongs to } \text{Pred}[S]\}$
2. Out sets: $\text{Out}[S] = \text{Generated set of } S \text{ union } (\text{Input set of } S \text{ minus Killed set of } S).$

Where:

- S is a basic segment in the CFG.
- $\text{Pred}[S]$ represents the set of predecessor basic segments of S .
- The generated set(S) indicates the set(definitions) generated by basic segment S .
- Killed set(S) indicates the set(definitions) killed by basic segment S .

CHAPTER 3

SOFTWARE REQUIREMENT SPECIFICATIONS

3.1 SCOPE

The study introduces a new method, the Topic Stability Driven Approach (TSDA), aiming to improve topic reliability in documents. It combines L-DA and L-SA, enhanced by Grid Search and Genetic Algorithm for better topic modeling. By checking coherence scores, it ensures consistent topics. It also includes sentiment analysis to understand emotional expressions, proving its effectiveness across different datasets. TSDA promises better topic sentiment summarization for practical use.

3.2 METHODOLOGIES

- Proposing the Topic Stability Driven Approach (TSDA) to enhance the stability and reliability of topic extraction from documents.
- Implementing L-DA and L-SA as primary topic modeling algorithms.
- Utilizing Grid Search and Genetic Algorithm for optimizing parameters of LDA and LSA models to improve their performance.
- Evaluating coherence score metrics for each document to ensure the stability and consistency of identified themes across different analyses.
- Incorporating sentiment analysis to understand the emotional expressions in text and analyzing sentiment variations across different topics.
- Refining the methods through hyperparameter optimization to ensure adaptability across various types of text and topics.
- Using coherence score metrics as a quantitative measure to assess the quality and coherence of topics extracted from documents.

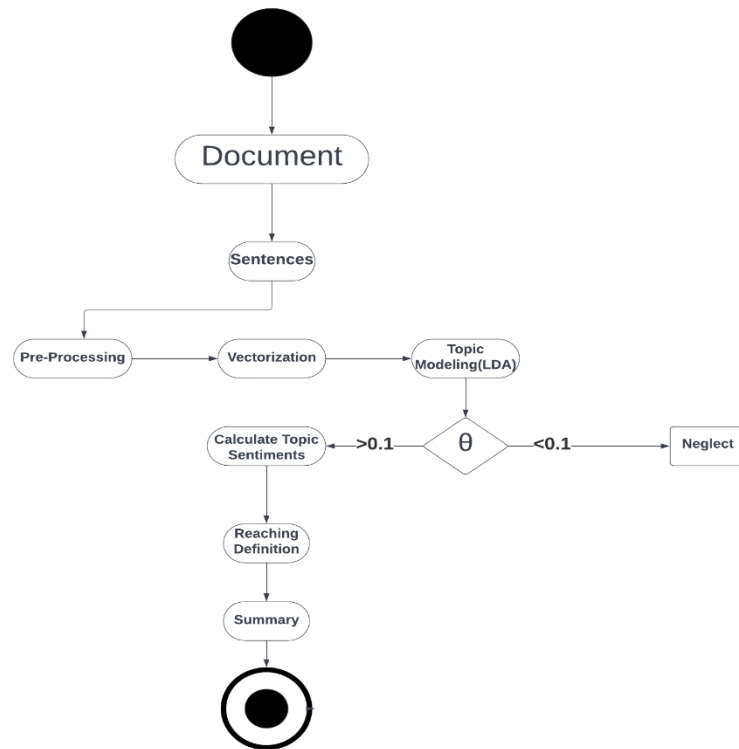


Figure 3.2.1 Activity Diagram of our project

3.3 SYSTEM REQUIREMENTS

The software and hardware requirements of the system are as follows:

3.3.1 HARDWARE INTERFACES

- Intel® Core™ i5-8265U 1.6GHz
- 8 GB RAM

3.3.2 SOFTWARE INTERFACES

- Jupyter Notebook
- Technologies used – Python
- Google Colab

3.3.2.1 JUPYTER NOTEBOOK

Jupyter Notebook is a tool for interactive work, allowing users to mix code, equations, and text. It supports Python, R, Julia, and more, making it useful for data analysis, research,

and learning. Its blend of code and text helps create understandable, shareable workflows for exploring data and collaborating. With features like plots and markdown, it promotes clear communication and reproducibility in research and analysis.

3.3.2.2 PYTHON

Python stands as a prevalent, accessible programming language acclaimed for its simplicity and legibility. It appeals to novices and experts alike, finding utility across a spectrum of endeavors including Digital Solutions Engineering. Python's robust library and seamless cross-platform adaptability render it an agile and proficient choice for software development across diverse computing environments.

3.3.2.3 GOOGLE COLAB

Google Colab is a free online platform for running Python code. It works like a shared notebook where you can write and run code together. You can use powerful processors like GPUs and TPUs for faster calculations, perfect for tasks like machine learning. It also connects smoothly with Google Drive for storing and sharing files. Plus, it makes installing libraries easy, simplifying project setup.

3.4 FUNCTIONAL REQUIREMENTS

3.4.1 FRQ1. Input Module

Description : Takes multiple text file and give as a single .csv file.

Functional Response: It iterates all text document in a particular directory and read each file and writes into a pandas data frame with specific index value and returns as a single .csv file.

3.4.2 FRQ2. Pre-Processing Module

Description : Converts raw data into usable data

Functional Response: The raw data can have special characters, html tags, stop words that doesn't have any meaning, numerical values such as date, time, rupee etc. which will affect the accuracy of model and makes the model inaccurate. Hence all these are removed from the text data.

3.4.3 FRQ3. Vectorization

Description : Converts preprocessed data into machine understandable format.

Functional Response: The machine cannot understand text other than 1's and 0's. so the preprocessed data is converted into a vector of unique words and their counts in a sentence or a document.

3.4.4 FRQ4. LDA Topic Modelling

Description : Generates topic-word distribution and sentence-topic distribution from the given input corpus or vector

Functional Response: LDA topic modeling generates topic-word distributions by assigning probabilities to words within topics based on their frequency. Similarly, it computes sentence-topic distributions by assigning probabilities to topics within sentences, reflecting the prevalence of topics in each sentence. This process involves iterative inference to optimize the distribution parameters, ultimately producing coherent topics and their corresponding word and sentence distributions.

3.4.5 FRQ5. Sentiment Embedding Calculation

Description : Generate Sentiment Embedding Using Sentic-net 6

Functional Response: Using the topics generated from LDA topic model. Sentiment Embeddings are calculated from the linear superposition of sentiment of each word from each topic using Sentic-net 6.

3.4.6 FRQ6. Document Structure Analysis

Description : Helps to identify the flow of sentiment in a document.

Functional Response: The system iteratively compares topics across different sentences to analyze the flow of sentiment. Sentiment flow is categorized into three structures: total-sub, sub-total, and circular. These structures provide insights into how sentiments evolve and interact throughout the document, facilitating a deeper understanding of sentiment dynamics.

3.4.7 FRQ7. Summarization Module

Description : Generates final topic summaries

Functional Response: The system utilizes the Summary Generation module to create final topic summaries. Sentiments within each node are monitored until they stabilize, indicating steady states determined by Reaching Definition. The output of the last node serves as the final topic sentiment summary.

3.4.8 FRQ8. Evaluation Module

Description : Evaluate the summary generated using summarization algorithm.

Functional Response: The system employs a Support Vector Machine (SVM) classifier with 5-fold cross-validation to evaluate the accuracy of summaries generated by the summarization algorithm.

3.4.9 FRQ9. Average Weighted Methods

Description : Recalculates the sentiment assignments.

Functional Response: The system recalculates assignments of variables by considering the sentiment weights of sentences and their positions. It implements static and dynamic weighted average methods for sentiment neutralization, and sentiment superposition to adjust new assignments based on the relative positions of sentences. These methods enhance the understanding of sentiment dynamics within the document.

3.4.10 FRQ10. Topic Sparsification

Description : Removal of unimportant topics

Functional Response: The system introduces a topic sparsification parameter sets it to 0.1. Topics with weights exceeding the value are retained, while those with weights below 0.1 are filtered out as unimportant. This filtering process helps prioritize significant topics, enhancing the relevance and quality of the generated summaries.

3.5 NON-FUNCTIONAL REQUIREMENTS

3.5.1 PERFORMANCE REQUIREMENTS

NFRQ1. Robustness: The topic sentiment summarization framework should be robust, capable of effectively handling variations in document content and sentiment expressions without compromising performance. It should adapt to changes in sentiment flow and topic distribution, ensuring reliable summarization outcomes across diverse datasets and domains.

NFRQ2. Quickness: The system's processing capabilities should be swift, enabling real-time analysis of textual inputs and prompt generation of summarized outputs. It should respond to user interactions without delay or buffering, ensuring a seamless and responsive user experience during the summarization process.

3.5.2 SCALABILITY REQUIREMENTS

NFRQ3. Scalability: The model should demonstrate high scalability, capable of accommodating increasing amounts of data or processing larger documents efficiently without compromising performance or stability.

3.5.3 SOFTWARE QUALITY ATTRIBUTES

NFRQ4. Maintainability: The system should be designed with a modular and well-documented structure to facilitate easy maintenance and updates. Codebase should adhere to best practices, enabling future modifications and enhancements.

NFRQ5. Testability: The design should be testable with different parameters.

NFRQ6. Adaptability: The framework should be adaptable to different datasets and domains, allowing users to apply it in diverse contexts such as movie reviews (IMDb) and product reviews (Amazon Electronics).

NFRQ7. Portability: The system should be portable across different computing environments and platforms, enabling users to deploy and utilize it seamlessly on various systems. This involves ensuring compatibility with common operating systems and providing clear installation instructions for easy setup and usage.

CHAPTER 4

SYSTEM DESIGN

4.1 OVERVIEW

This section presents the overview of the whole system. The Section 4.2 shows the system architecture Section 4.2 defines the main five modules.

4.2 OVERALL ARCHITECTURE

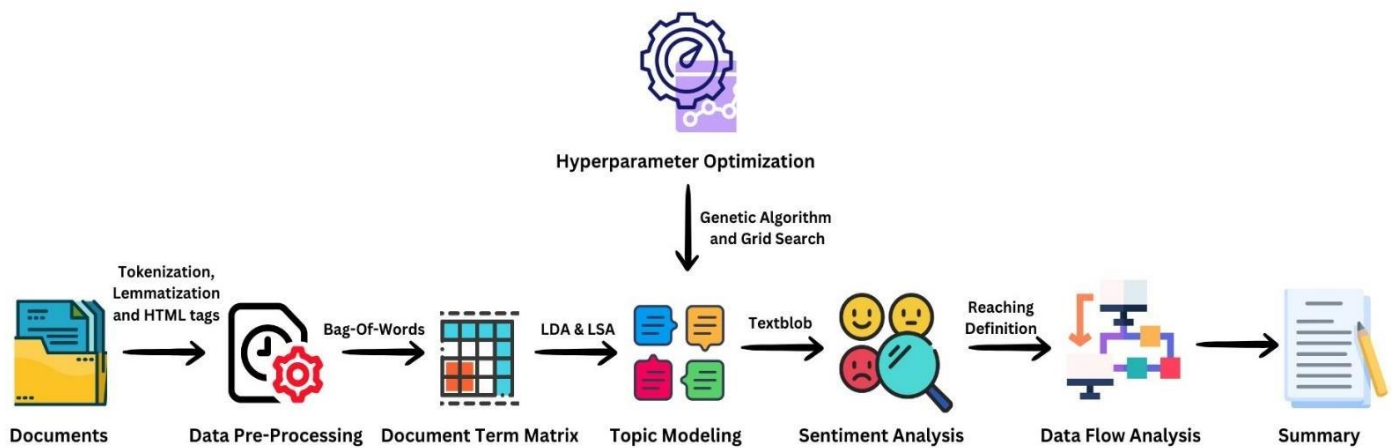


Figure 4.2.1 Overall Workflow of TSDA Framework

4.3 MODULES

- Data Preprocessing
- Document Term Matrix
- Topic Modeling
- Hyperparameter Optimization
- Sentiment Calculation
- Data Flow Analysis

4.3.1 DATA PREPROCESSING

Data cleaning plays a crucial role as a preprocessing step in NLP and text data analysis, serving to transform raw textual data into an organized format appropriate for significant evaluation. This process involves a series of meticulously designed procedures aimed at purifying the data by neglecting immaterial noise, standardizing formats, and improving the overall caliber of the textual information. Initially, the extraction of textual content from sources such as web-based HTML documents involves the removal of HTML tags, ensuring that only the actual text is retained for analysis. By eliminating these extraneous elements, the integrity and purity of the data are preserved, laying a solid foundation for subsequent analysis. Subsequently, the systematic removal of special characters, numeric digits, and punctuation marks further refines the text, streamlining its structure and enhancing readability. This step is instrumental in simplifying the textual data while streamlining content by removing non-essential elements that lack semantic significance. By standardizing the text in this manner, inconsistencies and irregularities are mitigated, facilitating more accurate analysis and interpretation. Additionally, data cleaning may involve tasks such as spell checking, stemming, and lemmatization, which further refine the text by reducing variations and normalizing word forms. Overall, data cleaning serves as a critical precursor to NLP and text analysis, laying the groundwork for subsequent processing and modeling tasks. By ensuring the quality and coherence of the textual data, data cleaning enhances the effectiveness and reliability of downstream analytical processes, ultimately enabling more insightful and meaningful insights to be derived from the text.

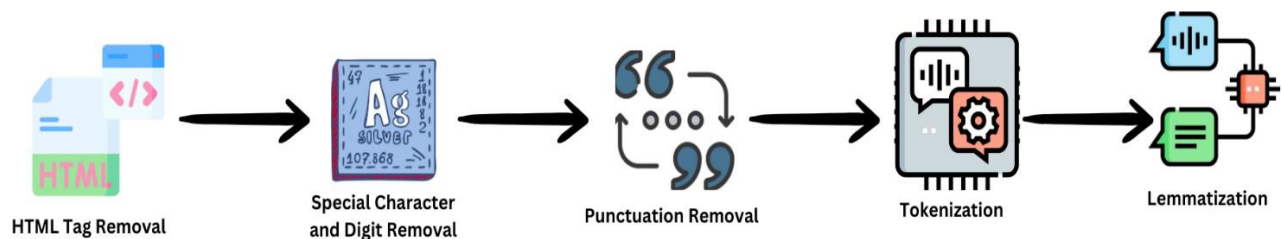


Figure 4.3.1.1 Steps involved in Data Preprocessing

4.3.2 DOCUMENT TERM MATRIX

The Document-Term Matrix (DTM) plays a fundamental role in text mining and natural language processing (NLP) by structuring textual data for analysis. It serves as a tabular representation where rows represent individual documents and columns represent unique words or terms across the entire corpus. This matrix provides a concise way to quantify the occurrence of terms within each document. To construct the DTM, the process typically begins with compiling a comprehensive list of unique words found in the entire collection of documents. This list forms the basis of what's commonly known as the dictionary or vocabulary.

Each word in this dictionary becomes a column in the DTM. Following the creation of the dictionary, each document in the corpus undergoes analysis using the BoW approach. In BoW, the order of words is disregarded, and only the frequency of each word is considered. This approach simplifies the representation of documents into vectors, where each component corresponds to the frequency of a term from the dictionary in the respective document.

Once the BoW representation is generated for each document, these vectors are assembled into the rows of the DTM. Each admittance in the matrix denotes the re-occurrence of a particular term in the corresponding text file. The resulting DTM provides a structured framework for conducting various analytical tasks such as text analytics, enabling researchers to uncover patterns, extract insights, and derive meaning from large collections of textual data.

4.3.3 TOPIC MODELING

Topic modeling serves as a method for uncovering the predominant themes or subjects within a compilation of documents. In our study, we use two common techniques for topic extraction: LDA and LSA.

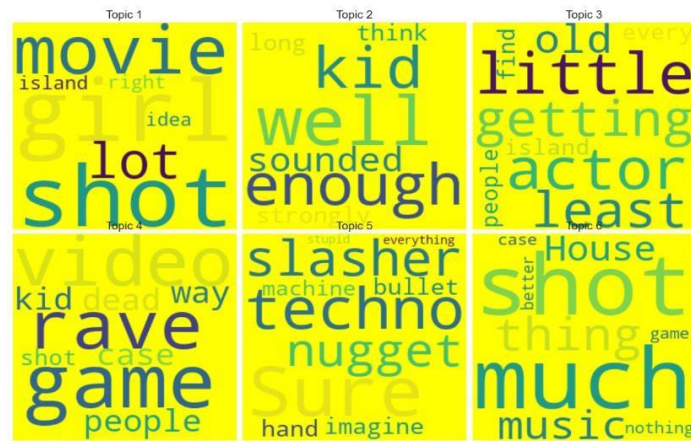


Figure 4.3.3.1 Word Cloud for Topics and Topic words

Latent Dirichlet Allocation (LDA) is a statistical model commonly used in natural language processing to uncover hidden topics within a collection of documents. The fundamental principle underlying LDA is the presumption that each text file in the corpus is composed of a combination of diverse topics, and each word within a text file is associated with one of these topics. LDA operates through an iterative process where it assigns words to topics and adjusts these assignments to better reflect the observed distribution of words across documents. By iteratively refining these assignments, LDA aims to reveal the underlying structure of topics within the corpus.

LSA is a computational method employed in NLP to extract and understand the underlying connections between words and text file within a huge corpus of textual data. It operates by scrutinizing the frequency of word occurrences across documents and represents them in a high-dimensional matrix. This matrix captures the co-occurrence patterns of words across different documents, forming the basis for semantic analysis. Central to LSA is the application of Singular Value Decomposition (SVD), a mathematical technique used to reduce the dimensionality of the matrix while preserving essential semantic correlations. Through SVD, LSA condenses the feature space of the matrix, thereby uncovering latent semantic structures inherent in the text data.

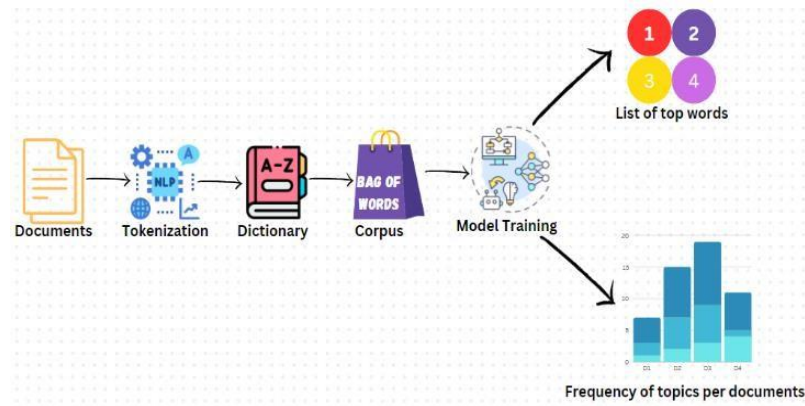


Figure 4.3.3.2 Topic modeling encompasses both LDA & LSA

4.3.4 HYPERPARAMETER OPTIMIZATION

Hyperparameter optimization is a crucial step in fine-tuning machine learning models to achieve optimal performance. In the context of our study, we focus on enhancing the effectiveness of our topic modeling method by adjusting specific settings known as hyperparameters. These hyperparameters act as controls, influencing how the model operates and performs. Analogous to dials on a machine, adjusting these hyperparameters enables us to optimize the model's functionality. In our research, we employ two primary techniques for hyperparameter optimization: Grid Search and Genetic Algorithm (GA). Grid Search is a systematic approach wherein we explore various combinations of hyperparameter values to identify the configuration that yields the best performance. It operates akin to experimenting with different recipes, where we adjust ingredient quantities to achieve the desired balance. By creating a grid of potential hyperparameter values and systematically testing each combination, Grid Search helps us identify the optimal settings for our topic modeling method.

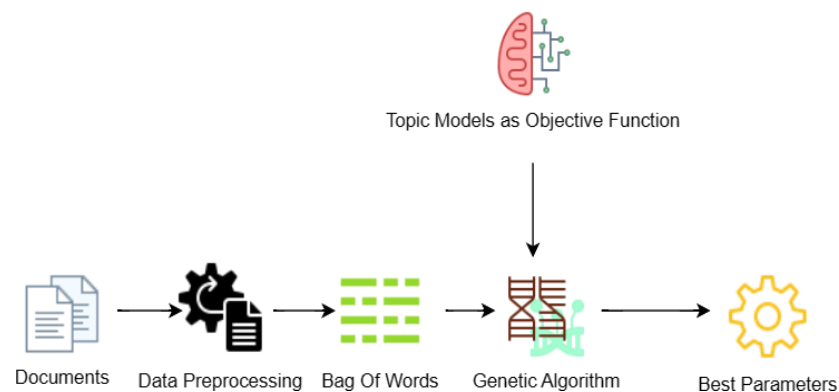


Figure 4.3.4.1 Finding Best Parameters for LDA Model

On the other hand, Genetic Algorithm mimics the process of natural selection, propelled by adaptive principles. It begins with a community of capable solutions, each symbolized by a set of hyperparameters. These solutions undergo evaluation based on their performance in the given task. The algorithm then selects the best-performing solutions and combines them to produce offspring solutions through crossover and mutation operations. This iterative process continues over multiple generations until the algorithm converges on the best combination of hyperparameters. Like evolution refining the traits of a species over time, Genetic Algorithm progressively improves the performance of our topic modeling method by selecting and refining the most promising hyperparameter configurations.

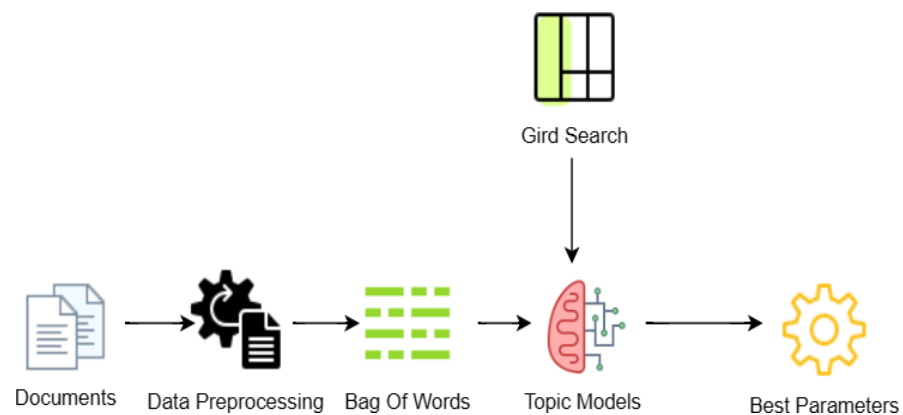


Figure 4.3.4.2 Finding Best Parameters for LSA Model

4.3.5 SENTIMENT CALCULATION

In this process, each sentence in a document is linked with topics generated from topic models based on the sentence's topic distribution. Any topics with a low relevance (less than or equal to 0.1) are discarded as unwanted. Then, the words in the sentences are compared with the words associated with the remaining topics. If there's a match, sentiments are combined for each sentence. Sentiments are determined using a dictionary called Text-Blob, which assigns sentiment scores to words based on their meaning. Essentially, this method helps in understanding the overall sentiment expressed in each sentence by considering the words' sentiments and their association with specific topics. Alternative dictionaries besides Text-Blob exist for SA, offering diverse approaches and lexicons tailored to specific contexts. One such option is VADER, known for its effectiveness in capturing sentiment nuances, especially in social media texts.

Senti WordNet, based on Word-Net syn-sets, assigns sentiment scores to words based on their syn-set's positivity, negativity, and neutrality. Another popular choice is A-FINN, a simple yet efficient dictionary-based method that assigns polarity scores to words. These alternatives provide flexibility and customization in SA, catering to varied needs and domains beyond what Text-Blob offers.

Require: sentences $S = \{s_1, s_2, \dots, s_n\}$; topic words $W = \{w_1, w_2, \dots, w_n\}$;

Ensure: sentiment scores for each sentences

1. Initialize an empty list to store sentiment scores
2. **for** each sentence s_i in sentences S **do**
3. Initialize sentiment_score = 0
4. **for** each word in word_tokenize(s_i) **do**
5. **if** word in topic words W **then**
6. Calculate sentiment score of the word (TextBlob)
7. Add sentiment score of the word to sentiment_score
8. Append sentiment_score to sentiment_scores
9. **end for**
10. return sentiment_scores

4.3.6 DATA FLOW ANALYSIS

The document analysis process simplifies text understanding by first checking each sentence for positive, negative, or neutral feelings. Then, it combines each sentence's main topic and emotion to understand the overall content and mood. This method helps in efficiently grasping key themes and sentiments expressed within the document. The process of document analysis starts by examining the sentiment of each sentence. Then, it applies a method called reaching definition to determine what each sentence says about a given topic and sentiment. This helps identify which definitions are no longer valid (killed) and which are newly introduced (generated) as the analysis progresses. Whenever the same topic appears again, whether with the same or a different sentiment, the previous definition is discarded, and a new one is generated. This iterative process continues across the entire document, allowing the computation of both IN and OUT sets, which represent the information flowing into and out of each sentence.

Eventually, the summary is extracted from the final OUT set, which captures the main themes and sentiments of the document. This approach ensures that the summary reflects the evolving understanding of the document, considering how topics and sentiments develop and change throughout its content. By continuously recalculating definitions and updating sets, the analysis stays current with the document's progression, providing a comprehensive overview that captures its essence effectively.

Require: Topic index of each sentence and corresponding sentiments

Ensure: IN and OUT sets

1. Initialize empty lists: `definitions_generated = []`, `definitions_killed = []`,
`IN_sets = []`, `OUT_sets = []`
2. Iterate over each row
3. Initialize empty lists: `row_definitions_generated = []`, `row_definitions_killed = []`
4. Process each topic index, and sentiment score in the row:
5. Define the current definition as `(topic_index, sentiment_score)`
6. Determine if any previous definitions match the current one and mark them as killed
7. Add the current definition to the generated set and merge with previous sets if not killed
8. Append generated and killed sets for the row to respective lists
9. Execute the reaching definitions algorithm to calculate IN and OUT set
10. Initialize OUT sets for the row
11. Iterate until convergence
12. Update IN and OUT sets for each sentence based on previous sets
13. Repeat until OUT sets no longer change
14. Append IN and OUT sets to their respective lists
15. return IN and OUT sets

4.3.7 SENTIMENT CLASSIFICATION

After generating summaries, we employ Text Blob to assess their sentiment, categorizing them as positivity, negativity, or neutrality. This sentiment analysis process is also applied to the original documents. By scrutinizing the sentiments of both the summaries and the original content, we gain an enhanced comprehension of their overall tone and emotions.

This method offers valuable perspectives into the sentiment conveyed within the text, enabling more comprehensive comprehension and analysis. Understanding whether the text exudes positivity, negativity, or neutrality aids in interpreting its underlying message and evaluating its impact. By systematically examining the sentiments of both the summaries and the original documents, we can discern patterns, identify key themes, and gauge the overall sentiment trajectory throughout the text. These perspectives provide a cornerstone for subsequent analysis, informing decision-making processes, and informing subsequent steps in data interpretation or decision-making. Overall, sentiment analysis using Text Blob enhances our ability to extract meaning from textual data, offering valuable context, and facilitating more informed conclusions.

IMPLEMENTATION METHODOLOGY

CHAPTER 5

IMPLEMENTATION METHODOLOGY

5.1 OVERVIEW

The implementation methodology describes the main functional requirements, which are needed for doing the project.

5.2 ESSENTIAL LIBRARIES

The libraries used in this project are Gensim, Text Blob, Truncated SVD, Wordnet Lemmatizer, Scikit Learn, NumPy , Pandas, Ast, Matplotlib, Seaborn, Beautiful Soup, NLTK, Word Cloud and Re.

5.2.1 GENSIM

Gensim is a Python library designed for Topic modeling, Text file categorization, and analogous retrieval with large corpora. It is particularly powerful for processing large text data and extracting semantic meaning from documents. Gensim provides implementations for popular algorithms such as LSA, LDA, and Word-Vec. These algorithms allow users to analyze text data, discover hidden patterns, and extract meaningful insights. Gensim also offers functionalities for similarity queries, text abridgment, and text file clustering, making it a most needed tool for NLP tasks.

5.2.2 TEXTBLOB

Text Blob is a simple and easy-to-use Python library for processing non-organized data. It provides a high-level interface for common-NLP tasks such as emotion findings, grammatical tagging, NP extraction, and language conversion. Text Blob wraps popular NLP libraries such as NLTK and Pattern, making it convenient for beginners and researchers alike. With Text Blob, users can perform sentiment analysis on text data to determine the overall sentiment frequency of a document or sentence.

5.2.3 TRUNCATED SVD

Truncated (SVD) Singular-Value-Decomposition is a space reduction strategy widely used in text analysis and information retrieval. It is particularly useful for reducing the dimensionality of sparse matrices, such as those encountered in text data represented using strategies like TF-IDF. Truncated SVD works by decomposing a matrix into three matrices: U , Σ , and V^T , and upholding only the peak k individual values and corresponding individual vectors. This condensed depiction preserves the most important information in the original matrix while discarding noise and reducing computational complexity. Truncated SVD is often employed in tasks such as text file clustering, latent semantic indexing, and Topic modeling to extract meaningful features from high-featured text data efficiently.

5.2.4 WORDNET LEMMATIZER

The WordNet Lemmatizer, available in the NLTK library, is a tool for performing lemmatization is a process of decreasing words to their foundation form, known as lemmas. For example, the words "running," "ran," and "runs" would all be lemmatized to the base form "run." WordNet Lemmatizer utilizes WordNet's extensive database of word forms and their corresponding lemmas to perform accurate and context-aware lemmatization, making it a valuable preprocessing step in many NLP tasks.

5.2.5 SCIKIT LEARN

Scikit-Learn is a comprehensive machine learning library for Python, providing simple and efficient tools for data analysis, modeling, and prediction. It offers a wide range of algorithms and utilities for various machine learning tasks, including classification, regression, clustering, dimensionality reduction, and model evaluation. Scikit-Learn is built on top of other popular scientific computing libraries such as NumPy, SciPy, and Matplotlib, and is designed with an intuitive and consistent API that makes it easy to use and integrate into existing workflows. With Scikit-Learn, users can preprocess data, train machine learning models, tune hyperparameters, and evaluate model performance using a unified interface.

5.2.6 SWIFTER

SWIFTER is a library designed to optimize the performance of Pandas operations, particularly when working with large datasets. It achieves this by automatically parallelizing operations, making use of all available CPU cores efficiently. This can significantly accelerate data processing tasks, particularly those involving complex computations or transformations on Data Frame objects.

5.2.7 AST (Abstract Syntax Tree)

AST is a built-in Python module that gives materials for parsing Python source code into an abstract syntax tree representation. This tree structure allows developers to analyze, manipulate, and understand the structure of Python code programmatically. It's commonly used in tools for code analysis, transformation, and optimization.

5.2.8 NLTK (Natural Language Toolkit)

NLTK stands as a comprehensive Python library tailored for addressing a spectrum of NLP tasks. Its arsenal encompasses a suite of materials and resources essential for diverse functions within the NLP domain, ranging from sentence or word splitting to grammatical tagging, and NER.

5.2.9 BEAUTIFUL SOUP

BS is a popular Python library for parsing markup files. It provides uncomplicated methods for navigating and extracting data from markup files, making it particularly useful for web scraping tasks. BS can handle poorly formatted markup and supports various parsers, allowing flexibility in dealing with different types of documents.

5.2.10 SEABORN

Seaborn represents a data representing toolkit constructed atop Matplotlib. It furnishes an elevated interface for crafting visually appealing and enlightening statistical graphics. Seaborn streamlines the task of producing intricate visualizations like categorical plots, regression plots, and heatmaps, employing succinct and articulate syntax. Furthermore, it provides integrated themes and color schemes to augment the visual appeal of graphics.

5.2.11 MATPLOTLIB

Matplotlib stands as a robust graphing toolkit within Python, highly favored for crafting static, interactive, and animated data illustrations. Its capabilities span a broad spectrum, encompassing the creation of line graphs, scatter plots, bar charts, histograms, and beyond. Matplotlib grants users meticulous command over plot components, facilitating tailored adjustments to elements like axes, labels, hues, and annotations.

5.2.12 NUMPY

NumPy stands as an indispensable cornerstone for numerical computations within Python. It furnishes extensive backing for arrays of multiple dimensions, alongside a diverse array of mathematical functionalities. The efficacy of NumPy's array operations coupled with its broadcasting prowess renders it indispensable for endeavors in scientific computation and data analytics.

5.2.13 WORDCLOUD

Word-Cloud is a library for generating word clouds from text data. It visualizes the frequency of words in a corpus by displaying them in a graphical form, with more frequent words appearing larger. Word-Cloud offers customization options for controlling the appearance of the word cloud, including font size, color scheme, and layout.

5.2.14 RE (REGULAR EXPRESSIONS)

The re module gives materials for working with regular expressions in Python. Regular expressions are patterns used for matching and manipulating text strings. The re module allows developers to search, extract, and replace text based on these patterns, enabling powerful text processing capabilities such as pattern matching, splitting, and substitution.

5.3 FUNCTIONS USED FOR IMPLEMENTATION

The user defined function used for the implementation of the project are

5.3.1 PREPROCESSING

The function `preprocess()` is a comprehensive text preprocessing tool designed to clean and normalize sentences for analysis. It employs a series of techniques including HTML tag removal, contraction expansion, tokenization, stop words removal, digit removal, repeated character reduction, curly bracket removal, special character removal, ellipsis handling, word length filtering, and lemmatization. This ensures that the processed sentences are standardized and ready for further text analysis tasks.

5.3.2 SENTIMENT CLASSIFICATION

The function `"classify_sentiment()"` has been devised to ascertain the sentiment frequency of provided textual file data, leveraging the capabilities of the Text-Blob library. Text-Blob, a Python resource for textual file data processing. Within this function framework, sentiment frequency denotes the frequency of positivity or negativity disclosed in the text. A score of -1 suggests adverse sentiment, while 1 shows highly good sentiment, with 0 illustrating neutrality. Initially, the function accepts input text and directs it to Text Blob's sentiment analysis module. Text-Blob employs an array of NLP strategies to evaluate textual sentiment and derive its polarity. Subsequently, the function scrutinizes the polarity value to discern the overarching sentiment of the text. Should the polarity surpass 0, the function infers a positive sentiment; conversely, a polarity below 0 implies a negative sentiment. In the event of a polarity equivalent to 0, signifying neutral sentiment, the function returns 'neutral'.

5.3.3 GENETIC ALGORITHM

The function `optimize_lsa_with_genetic_algorithm()` is designed to fine-tune the parameters of a LSA model using a GA. LSA is a technique used in NLP and data recovery to depict connections between a set of text files and the terms or words they contain. To optimize the LSA model, the function defines an evaluation function ``eval_func()`` that computes the fitness of a given solution, which in this case is the number of components for LSA. The fitness score is typically based on how well the LSA model performs its intended task, such as capturing the underlying semantic frame of collection of text files. The genetic algorithm is then employed to search for the optimal number of components by maximizing the fitness score over a predefined range. GA are heuristic search algorithms admired by the process of adaptive selection. They iteratively improve candidate solutions by applying operators such as selection, crossover, and mutation to generate new offspring. The parameters of the genetic algorithm, such as population size and number of generations, are configured to control the search process. By iteratively evaluating and evolving candidate solutions, the genetic algorithm aims to find an potential or nearly possible to be potential configuration of parameters for the LSA model, thereby enhancing its performance in capturing semantic relationships within the document collection.

5.3.4 GRID SEARCH

The function `optimize_lsa_with_grid_search()` is a utility designed to optimize Latent Semantic Analysis (LSA) by determining the optimal number of components through grid search. To begin, the function preprocesses the text data, which typically involves tasks such as tokenization, lowercasing, and removing stopwords and punctuation. This data furnishing step ensures that the text is in a required representation of structure for analysis. Next, the function counts the occurrence of each word in the preprocessed text. This step results in a numerical representation of the text data, often referred to as a DTM, where rows represent text file and columns represent terms, with cell values indicating the frequency of each word in each document. The core functionality of the function involves training multiple LSA models with different numbers of components. This process is facilitated by grid search, where the function systematically tries different options for the number of components and evaluates each model's performance using a specified criterion, such as maximizing variance explained or minimizing reconstruction error.

Finally, the function identifies the LSA model with the optimal number of components based on the chosen criterion and returns it for further analysis.

5.3.5 REACHING DEFINITION

The function `calculate_reaching_definition()` processes a DataFrame containing text data along with associated sentiment scores and topic indices. It computes two crucial aspects for each row: definitions generated and definitions killed. Definitions generated are identified by iterating through each sentence in the processed data, creating sets to store these definitions. It checks for any definitions killed by previous iterations and updates the generated definitions accordingly, ensuring no redundant additions. Definitions killed are determined by comparing the current definition with previous ones, marking any matches as killed. After processing each row, the generated and killed definitions are stored in separate lists, which are then assigned to new columns in the DataFrame. Essentially, this function facilitates the analysis of textual data by identifying the definitions of topics and sentiment scores across different sentences, akin to the concept of reaching definitions in compiler theory applied to text analysis.

5.3.6 SUMMARY

The function `generate_summary()` is designed to produce summaries from a set of input sentences, their corresponding indices, and sentiment scores. It operates by iterating through each document in the provided data, creating an empty string to accumulate the summary and a set to track the indices already added. Within each document, the function traverses the sets of indices, ensuring they fall within the document's bounds and have valid sentiment scores. If an index is both valid and has not been previously included in the summary, the corresponding sentence is appended to the summary string, and the index is recorded as added. Once all relevant indices for a document have been processed, the generated summary is appended to the output list. This approach ensures that each sentence is included in the summary only once, preventing redundancy. Overall, the function offers a straightforward method for text summarization, leveraging provided indices and sentiment scores to produce concise and informative summaries of the input documents.

5.4 FRAMEWORKS

TSDA comprises eight frameworks: LDA Nexus, LSA Evolution, TopicBurst LDA, TopicBurst LSA, GenAlgo LDA, GenAlgo LSA, Gridify LDA, and Gridify LSA.

LDA Nexus and LSA Evolution utilize iterative methods, evaluating coherence for each document across 5 to 12 topics in their corresponding models (LDA and LSA). The model achieving the highest coherence score is chosen, and its topics, topic words, and sentence topic distributions are then utilized for subsequent analysis and processing. This iterative approach ensures that the selected model captures the most coherent and relevant topics within the dataset.

TopicBurst LDA and TopicBurst LSA similarly employ iterative strategies, aiming to identify the most suitable number of topics for the entire dataset. They achieve this by computing coherence scores across a range of potential topic numbers and then averaging these scores to determine the optimal configuration. This iterative process ensures that the selected number of topics maximizes coherence across the dataset.

On the other hand, GenAlgo LDA and GenAlgo LSA employ a genetic algorithm methodology. They first preprocess and vectorize all documents, then feed them into LDA or LSA topic models. These models aim to optimize various parameters such as alpha, eta, decay, offset, and minimum probability for LDA, and the number of topics for LSA. Through this approach, the genetic algorithm iteratively refines the parameters to enhance the performance of the topic models, ensuring more accurate and effective topic analysis.

Gridify LDA utilizes a grid search technique to optimize parameters like the number of topics, random state, alpha, and eta. On the other hand, Gridify LSA focuses particularly on determining the ideal number of topics within the scope of 5 to 15. These frameworks operate distinctively to enhance topic sentiment summarization by exploring predefined parameter spaces and adapting to various dataset characteristics, showcasing their versatility and effectiveness in optimizing topic modeling processes.

PERFORMANCE METRICS

CHAPTER 6

PERFORMANCE METRICS

6.1 OVERVIEW

Evaluation of a summary can be approached through diverse methodologies to ascertain its effectiveness. These methods include direct confusion matrix analysis, which assesses the summary's accuracy in capturing important information. Rouge 1, 2, and L Scores assess how closely a summary matches a reference text using n-grams. Fluency & Relevance score, which evaluates the summary's readability and relevance to topics extracted. stratified cross-validation, a technique that ensures the robustness of the evaluation process by validating against various subsets of the data. Each method offers unique insights into different aspects of summary quality, collectively providing a comprehensive evaluation framework.

6.2 DIRECT CONFUSION MATRIX

In sentiment analysis, the direct confusion matrix serves as a pivotal tool for comparing sentiment labels between the summary and the original document. This matrix allows for a precise evaluation of the accuracy of sentiment classification. Accuracy, a fundamental metric computed from the confusion matrix, offers valuable insights into the alignment between the sentiments conveyed in the summary and those present in the original document. By quantifying the number of correctly classified sentiments, accuracy provides a clear indication of the degree to which the summary reflects the emotional nuances and overall essence of the original content. It essentially serves as a yardstick for assessing the fidelity of the summary's portrayal of sentiments. Through this evaluation, analysts can discern whether the summary effectively captures the intended emotions and conveys the primary message encapsulated in the original document. The computation of accuracy involves comparing the correctly classified sentiments with the total sentiments present in both the summary and the original text. This computation elucidates the extent to which the summary faithfully represents the sentiment distribution of the original document. Thus, accuracy emerges as a crucial measure in gauging the effectiveness and reliability of the summarization process, offering valuable feedback for refining techniques and enhancing the quality of automated summarization systems. Below is the formula for computing accuracy (A).

$$A = \frac{TrPos+TrNeg}{TrPos+TrNeg+FsPos+FsNeg} \quad (1)$$

6.3 RELEVANCY AND FLUENCY

Relevance evaluation in summarization aims to gauge the extent to which a summary encapsulates the essential themes of a document. This assessment relies on identifying key words extracted from the top-k words associated with N topics derived from the original document. By examining whether these significant topic words are present in the summary, analysts can ascertain their efficacy in capturing the primary ideas. The evaluation process essentially involves checking the presence of these needed topic words in the summary against their prominence in the original text. This approach enables quantitative measurement of how well the summary represents the core concepts and crucial information contained in the document. Consequently, it provides valuable feedback on the summarization process's success in distilling the essential content while maintaining relevance and fidelity to the original material. A simple formula for this evaluation could be:

$$R = \frac{\text{Number of Top-k Topic Words in Summary}}{\text{Number of Top-k Topic Words}} \times 100 \quad (2)$$

6.4 ROUGE SCORE

To evaluate the effectiveness of a summary, Rouge scores, specifically Rouge-1, Rouge-2, and Rouge-L, are commonly employed. These scores measure the similarity between the generated summary and a reference summary generated through a PageRank algorithm. Rouge scores assess the convergence of N-Grams, which are series of N words, between the two abridgments. A higher Rouge score suggests a greater likeness and coherence between the summary and the original document. By comparing N-Grams shared by the generated abridgment and the reference text, Rouge scores provide a metric of how well the concise text captures the essential information and structure of the original content. These scores are particularly useful for assessing the performance of automatic abridgment algorithms. To compute the average Rouge scores across multiple documents, specific formulas are used, which consider the Rouge scores of individual documents and aggregate them to provide an overall assessment of summarization quality.

$$\text{Avg R-1} = \frac{\sum_{i=1}^n R-1_i}{n} \quad (3)$$

$$\text{Avg R-2} = \frac{\sum_{i=1}^n R-2_i}{n} \quad (4)$$

$$\text{Avg R-L} = \frac{\sum_{i=1}^n R-L_i}{n} \quad (5)$$

6.5 COHERENCE SCORE

The coherence score is a metric used to evaluate the quality of themes generated by theme extraction algorithms, such as LDA or NMF. It measures how semantically related the words within a topic are, indicating the degree to which the topic is coherent or understandable. There are several methods for computing coherence scores, but one common approach is pointwise mutual information (PMI). PMI measures the likelihood of two words co-occurring together in each context, compared to what would be expected if they were independent. Higher PMI values indicate stronger associations between words, suggesting greater coherence. To calculate coherence scores, the algorithm typically considers each topic individually. It extracts the top N words from the topic and then computes pairwise PMI scores for all combinations of these words. These PMI scores are then aggregated using a chosen aggregation function, such as averaging or summing, to produce a single coherence score for the topic. Higher coherence scores indicate topics where the words are more closely related, suggesting a clearer and more coherent theme. Topic models can then be fine-tuned based on coherence scores to produce more meaningful and interpretable results. Coherence scores are essential for evaluating the performance of topic modeling algorithms and for selecting the optimal number of topics. They provide a quantitative measure of the quality of topics generated, helping researchers and practitioners assess the relevance and coherence of the extracted topics.

RESULTS AND DISCUSSION

CHAPTER 7

RESULTS AND DISCUSSION

7.1 OVERVIEW

This chapter explains the result of our project and the screenshots for each step are included and explained.

7.2 DATASETS

The datasets used in our projects are:

7.2.1 STATIONARY DATASET

The IMDb movie review dataset, an integral part of TSDA, comprises 12,500 reviews expressing both positive and negative sentiments. These reviews serve as crucial data samples for sentiment analysis within the TSDA framework, enabling comprehensive analysis and optimization techniques to enhance topic sentiment summarization in movie reviews.

The table below outlines the number of samples employed for each framework within the TSDA methodology.

TSDA Frameworks	Number of Data Samples
LDA Nexus	500
LSA Evolution	500
Topic Burst LDA	7500
Topic Burst LSA	7500
GenAlgo LDA	2500
GenAlgo LSA	7500
Gridify LDA	2500
Gridify LSA	7500

Table 7.2.1.1 Data Samples used for each frameworks.

7.3 SCREENSHOTS

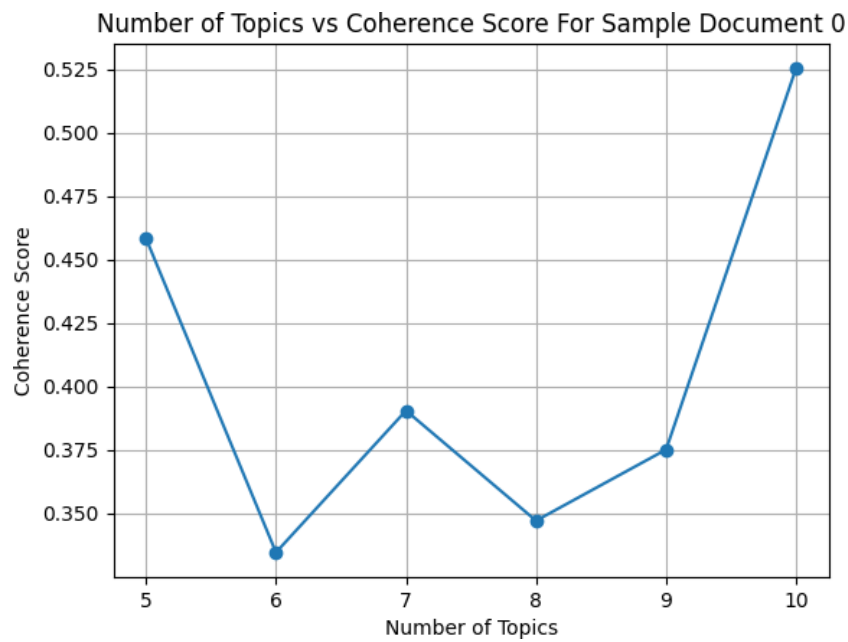


Figure 7.3.1 LDA Nexus – Topic Stability of a document.

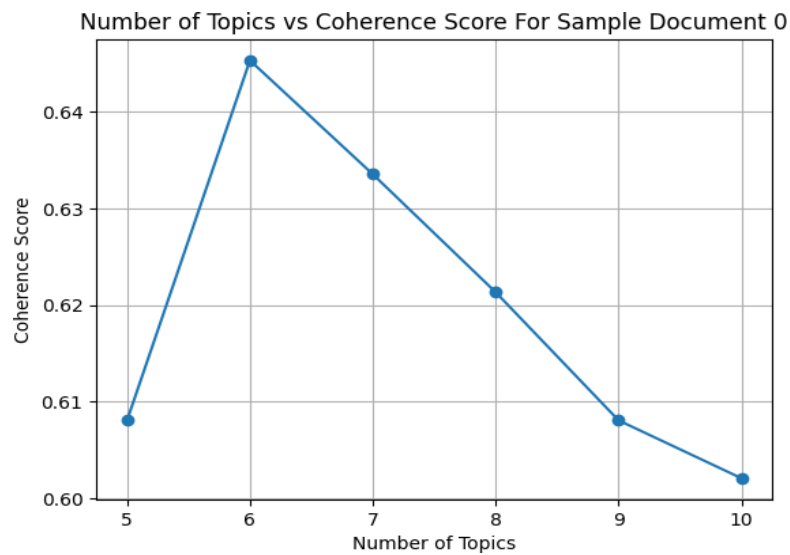


Figure 7.3.2 LSA Nexus – Topic Stability of a document.

Figures 7.3.1 and 7.3.2 illustrate the topic stability of a document as assessed by the LDA Nexus and LSA Evaluation Framework, respectively. These visual representations depict the consistency and reliability of topics identified within the document.

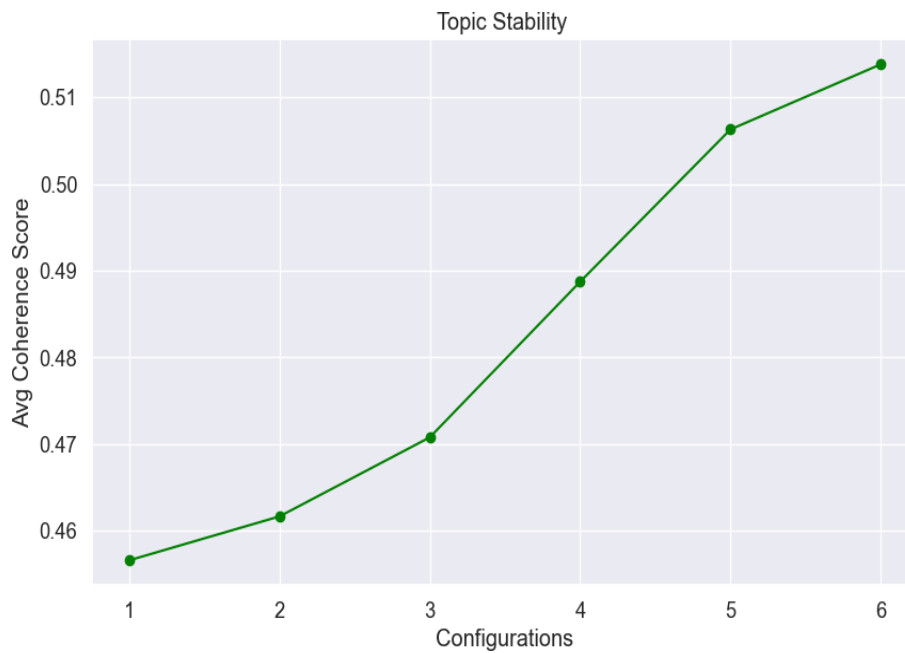


Figure 7.3.3 GenAlgo LDA – Average coherence score vs GA Configurations

Figure 7.3.3 displays the average coherence score compared to different configurations in GenAlgo LDA. This graph helps understand how changes in configurations affect the coherence score, aiding in the optimization of Genetic Algorithm parameters for LDA topic modeling.

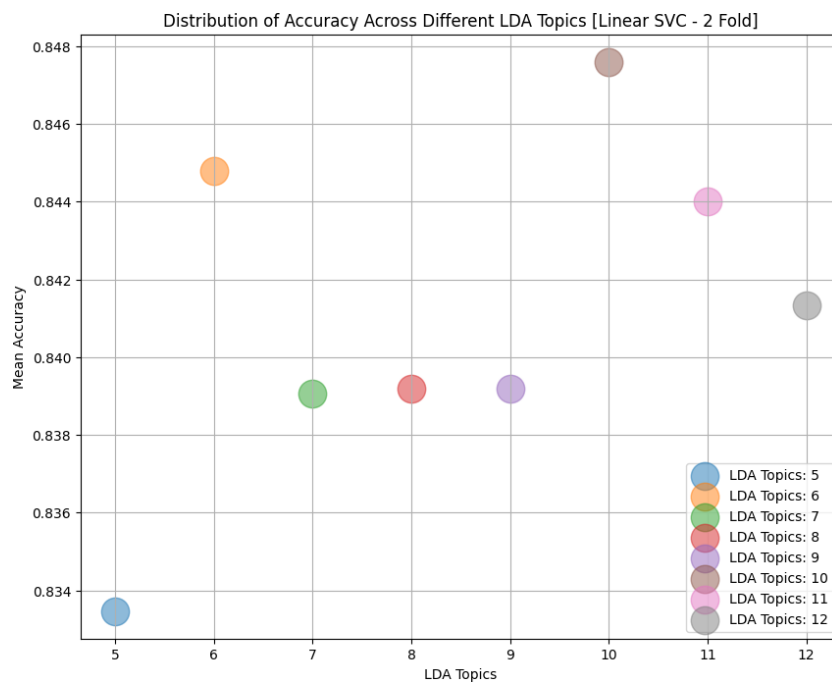


Figure 7.3.4 LDA Topics vs Mean Accuracy.



Figure 7.3.5 TopicBurst LDA – LDA Topics vs Accuracy using Direct CM Analysis.

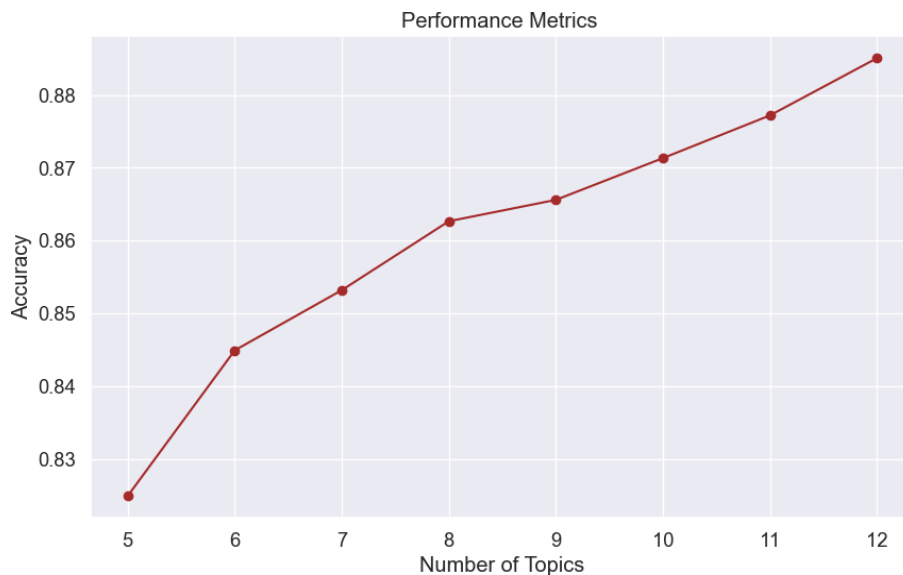


Figure 7.3.6 TopicBurst LSA – LSA Topics vs Accuracy using Direct CM Analysis.

Figures 7.3.5 and 7.3.6 illustrate accuracy compared to the number of topics generated by LDA and LSA topic models within the TopicBurst LDA and LSA frameworks. Through direct analysis of confusion matrices, these figures reveal that both frameworks demonstrate strong performance when employing 12 topics. This suggests that the accuracy of topic modeling within these framework's peaks at 12 topics, highlighting the effectiveness of this configuration in accurately summarizing topics within the given datasets.

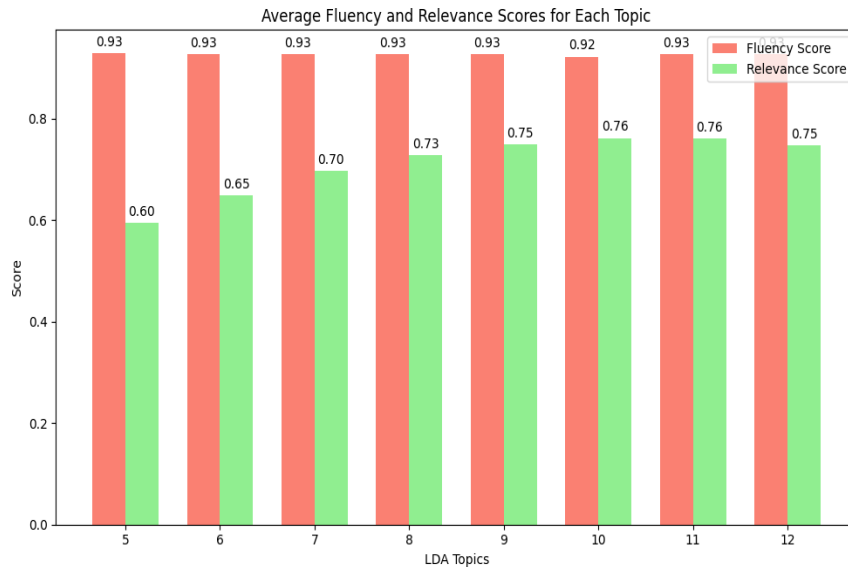


Figure 7.3.7 TopicBurst LDA – LDA Topics vs Fluency and Relevance Score.

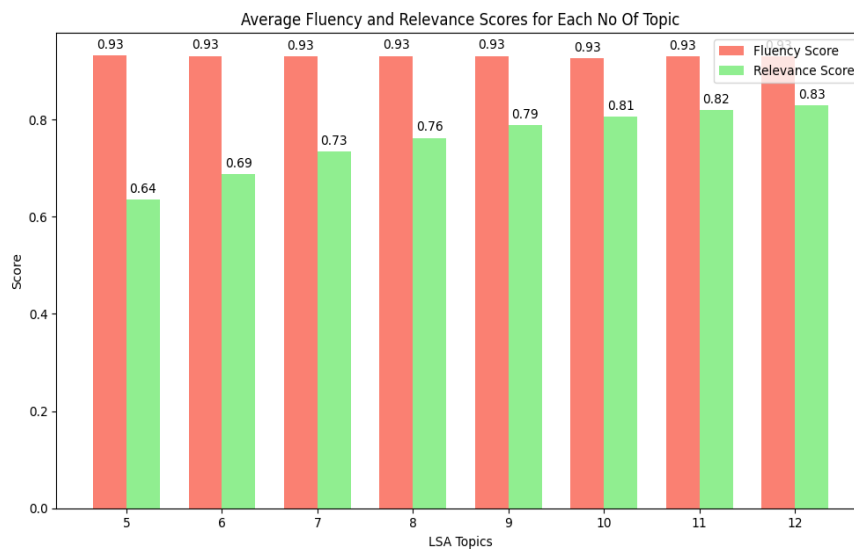


Figure 7.3.8 TopicBurst LSA – LSA Topics vs Fluency and Relevance Score.

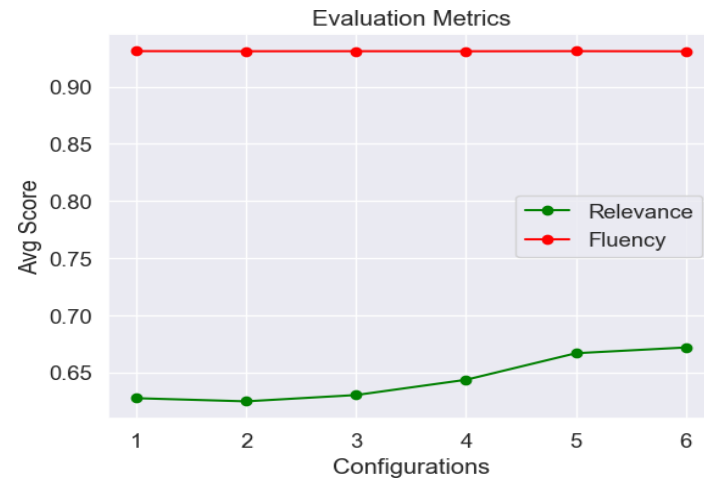


Figure 7.3.9 GenAlgo LDA – Configurations vs Fluency and Relevance Score.

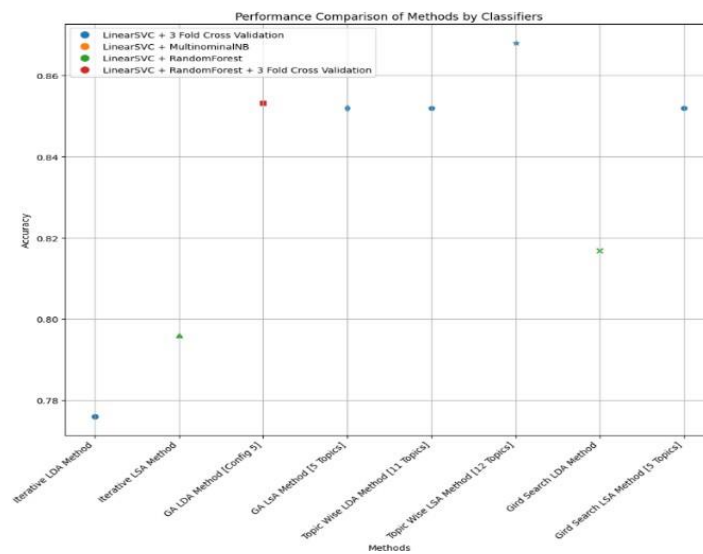


Figure 7.3.10 ML and Cross Validation vs Frameworks with and without Ensemble of models with best settings.

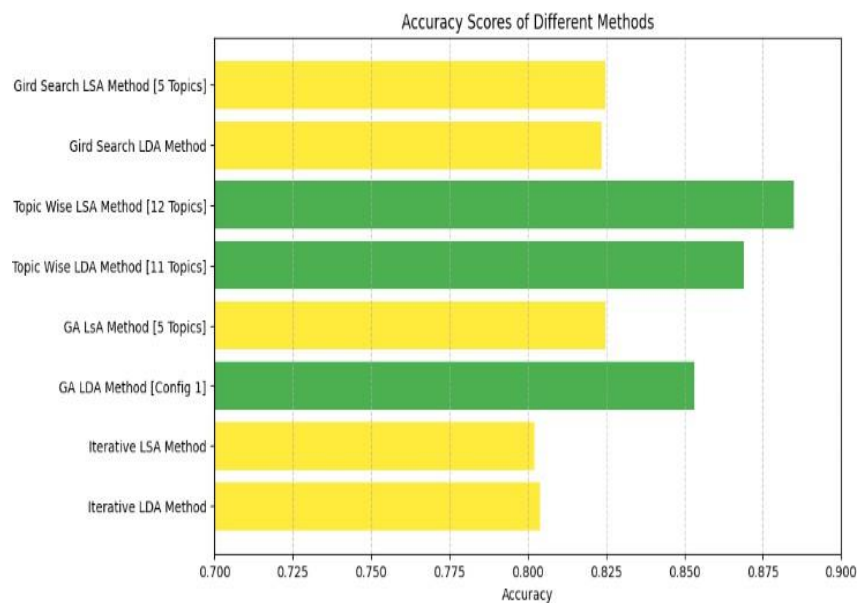


Figure 7.3.11 Accuracy vs Frameworks with best settings.

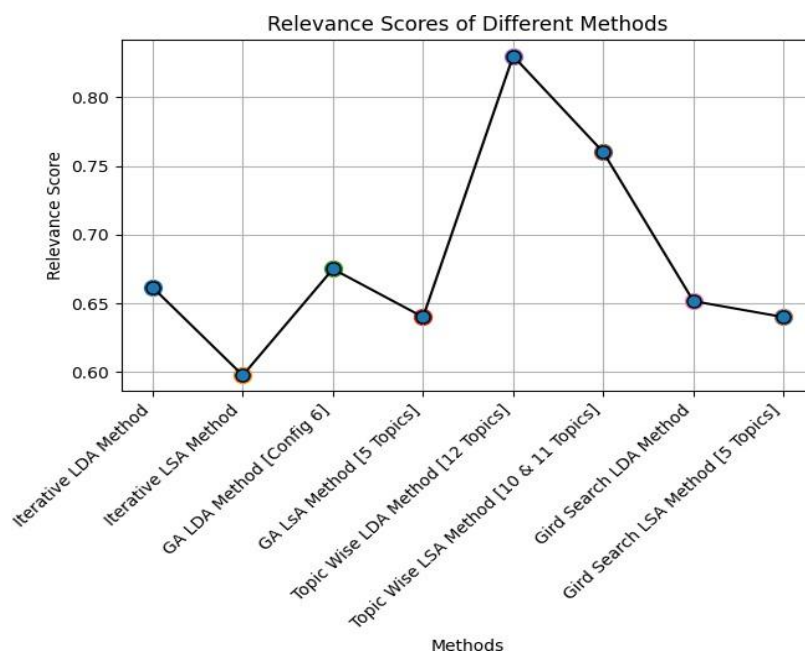


Figure 7.3.12 Relevance Score vs Frameworks with best settings.

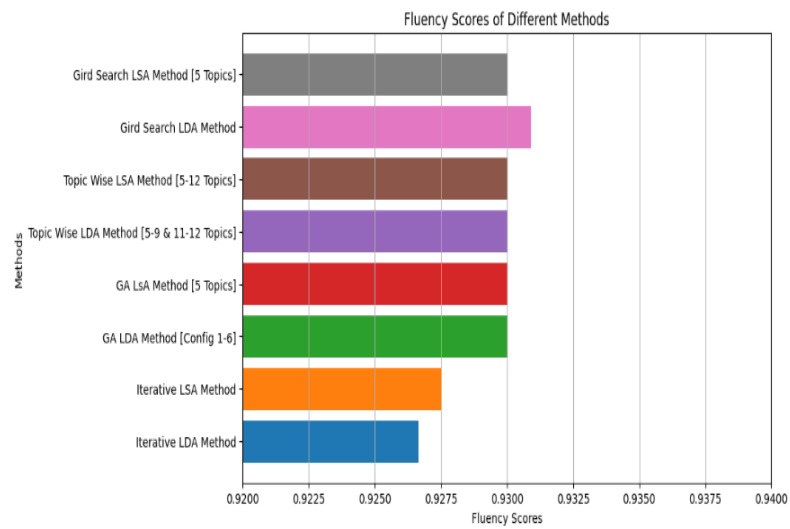


Figure 7.3.13 Fluency Scores vs Frameworks with best settings

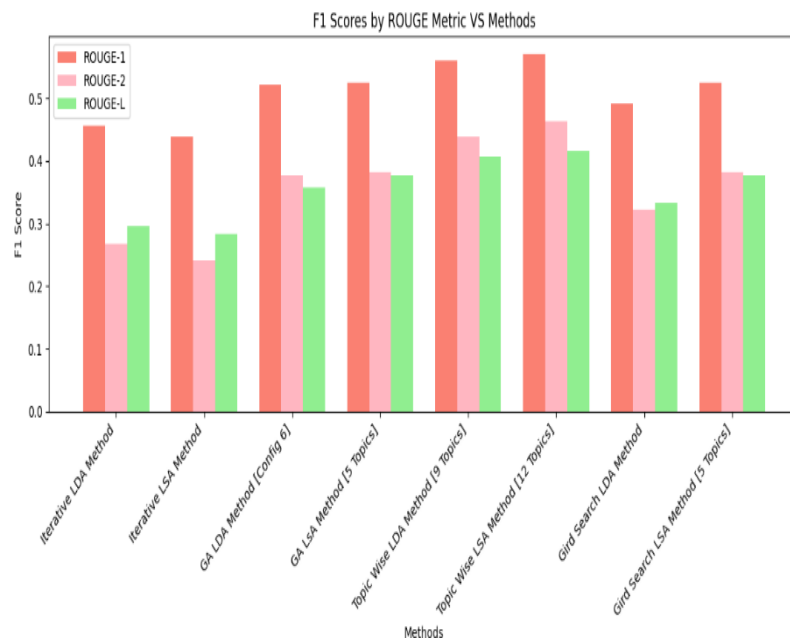


Figure 7.3.14 Rouge 1,2, L Scores vs Frameworks.

No of Topics	Alpha	Eta	Decay	Random State
5	0.5	0.1	0.5	42

Table 7.3.15 Best Parameters for LDA Model from Grid Search

Configurations	No of Topics	Alpha	Eta	Decay	Offset	Random State	Min Probability
1	5	0.11333	0.1802	0.6879	2.2924	23	0.0547
2	6	0.10841	0.4517	0.7843	4.8175	49	0.2030
3	7	0.10744	0.7574	0.7312	5.1262	9	0.0536
4	8	0.10874	0.2950	0.6020	9.9071	77	0.0853
5	9	0.11246	0.2876	0.6457	2.3621	86	0.0802
6	10	0.10250	0.9940	0.8194	9.3594	51	0.0102

Table 7.3.16 Best Configurations of Parameters for No of Topics from Genetic Algorithm for LDA Model.

CONCLUSION AND FUTURE WORK

CHAPTER 8

CONCLUSION

8.1 CONCLUSION

Across various evaluation metrics, several noteworthy observations emerge from the comprehensive analysis. Firstly, when considering accuracy measured through confusion matrix analysis, both Topic Wise LDA and LSA Methods, particularly with 11 and 12 topics respectively, exhibit notably high levels of accuracy, surpassing other methodologies. This suggests that these specific approaches excel in accurately classifying sentiments, indicating their efficacy in capturing the nuances of topics and sentiments within the text. Secondly, regarding fluency, all employed methods consistently achieve high scores, indicating a consistent level of linguistic quality across different frameworks. This uniformity suggests that regardless of the specific method utilized, the resulting summaries maintain a high level of readability and coherence. However, when evaluating relevance, the Topic Burst LDA Method with 12 topics stands out with the highest score, indicating its effectiveness in capturing and summarizing relevant information from the original text. Similarly, when assessed through Rouge scores, the Topic Wise LSA Method with 12 topics consistently demonstrates superior performance across Rouge 1, 2, and L metrics, underscoring its proficiency in summarization tasks by effectively capturing key information and maintaining coherence with the source material. Notably, in classification tasks, models employing Genetic Algorithm (GA) with various configurations exhibit competitive accuracy rates, implying the effectiveness of optimization techniques in improving classification outcomes. Lastly, in relevance assessment, the Topic Burst LDA Method once again emerges as the top performer, emphasizing its capability in generating summaries that are deemed highly relevant to the original text. Overall, the Topic Burst LDA Method with 12 topics emerges as the most effective approach across multiple evaluation metrics, demonstrating its superiority in topic sentiment summarization tasks by achieving high levels of accuracy, fluency, relevance, and classification performance. These findings underscore the importance of selecting appropriate methodologies tailored to specific summarization objectives and highlight the potential of advanced techniques such as Topic Burst LDA and Genetic Algorithm optimization in enhancing the quality of automated summarization processes.

8.2 FUTURE WORK

In the future, TSDA aims to explore additional techniques beyond reaching definition analysis to further enhance document summarization. This task may entail incorporating sophisticated algorithms or methodologies to enhance the efficacy and precision of the summarization procedure. Additionally, there is a focus on optimizing the Genetic Algorithm used within TSDA. This optimization effort includes identifying the best parameters such as population size and crossover types to enhance the algorithm's performance. Furthermore, TSDA seeks to enhance sentiment calculation and classification capabilities. This could involve refining existing sentiment analysis techniques or integrating new approaches to better understand and interpret sentiment within documents.

APPENDIX

APPENDIX

9.1 CODING:

#preprocessing

```
def preprocess(sentences, min_word_length=2):
    preprocessed_sentences = []
    for sentence in sentences:
        s = BeautifulSoup(sentence, 'html.parser')
        sent = s.get_text()
        sent = contractions.fix(sent)
        w = word_tokenize(sent)
        stp = set(stopwords.words('english'))
        w = [wd for wd in words if wd.lower() not in stp]
        w = [re.sub(r'\d+', '', wd) for wd in w]
        w = [re.sub(r'\d', '', wd) for wd in w]
        w = [re.sub(r'(\.|\!|\?|\,)', '', wd) for wd in w]
        w = [re.sub(r'\{.*?\}', '', wd) for wd in w]
        w = [re.sub(r'^[\\s]*|[^\w\s\']-', '', wd) for wd in w]
        w = [re.sub(r'\.{3}', '', wd) for wd in w]
        w = [re.sub(r'\.{4}', '', wd) for wd in w]
        w = [re.sub(r'\.( +)', '.', wd) for wd in w]
        w = [wd for wd in w if len(word) >= min_word_length and not wd.isnumeric() and wd not in
string.punctuation]
        lemmat = WordNetLemmatizer()
        words = [lemmat.lemmatize(wd) for w in w]
        pre = ' '.join(words)
        preprocessed_sentences.append(pre)

    return preprocessed_sentences
```

#textblob

```
def classify_sentiFre(t):
    b = TextBlob(t)
    fre = b.sentiment.polarity
    if fre > 0:
        return 'pos'
    elif fre < 0:
        return 'neg'
    else:
        return 'neu'
```

#Genetic Algorithm

```
def optimize_lsa_with_genetic_algorithm(df, n_components_range=(5, 11), pop_size=50, n_generations=10):
    def eval_func(solution):
        n_components = int(solution[0])
        tokenized_texts = preprocess_data(df)
        lsa_model, vectorizer = train_lsa_model(tokenized_texts, n_components=n_components)
```

```

    fitness = evaluate_lsa_model(lsa_model, vectorizer, tokenized_texts)
    return fitness
vb = np.array([n_components_range])
parama = {'max_num_iteration': n_generations, 'population_size': pop_size}
md = ga(function=eval_func, dimension=1, variable_type='int', variable_boundaries=vb)
md.run()

best_solution = model.output_dict['variable']
best_n_components = int(best_solution[0])
tokenized_texts = preprocess_data(df)
final_lsa_model, final_vectorizer = train_lsa_model(tokenized_texts, n_components=best_n_components)
return final_lsa_model, final_vectorizer

```

#Grid Search

```

def optimize_lsa_with_grid_search(df, n_components_range=(5, 15), cv=5):
    tokenized_texts = preprocess_data(df)
    param_grid = {'n_components': range(n_components_range[0], n_components_range[1] + 1)}
    vectorizer = CountVectorizer()
    X = vectorizer.fit_transform(tokenized_texts)
    custom_scoring = make_scorer(custom_scorer, needs_proba=False, needs_threshold=False)
    gs = GridSearchCV(TruncatedSVD(), param_grid, cv=cv, scoring=custom_scoring)
    gs.fit(X)
    best_n_components = grid_search.best_params_['n_components']
    final_lsa_model, final_vectorizer = train_lsa_model(tokenized_texts, n_components=best_n_components)

    return final_lsa_model, final_vectorizer

```

#Rogue Score

```

def calculate_rouge(ref_sum, gen_sum):
    s = rouge_scorer.RougeScorer(['rouge1', 'rouge2', 'rougeL'], use_stemmer=True)
    sc = s.score(ref_sum, gen_sum)
    r1_s = [sc['rouge1'].precision, sc['rouge1'].recall, sc['rouge1'].fmeasure]
    r2_s = [sc['rouge2'].precision, sc['rouge2'].recall, sc['rouge2'].fmeasure]
    rL_s = [sc['rougeL'].precision, sc['rougeL'].recall, sc['rougeL'].fmeasure]

    return r1_s, r2_s, rL_s

```

#Relevance Score

```

def evaluate_relevance(summary, topic_keywords):
    relevance_score = 0
    if isinstance(summary, str):
        summary_lower = summary.lower()
    else:
        return relevance_score
    if isinstance(topic_keywords, list):
        for keyword in topic_keywords:
            if isinstance(keyword, str):

```

```

        if keyword.lower() in summary_lower:
            relevance_score += 1
        else:
            continue
    if len(topic_keywords) > 0:
        normalized_relevance_score = relevance_score / len(topic_keywords)
    else:
        normalized_relevance_score = 0
    return normalized_relevance_score

# filter topic distribution
import ast
import swifter
def filter_topic_distribution(topic_distribution_list):
    theta = 0.1
    t1_list = []
    distribution_list = ast.literal_eval(topic_distribution_list)
    for distribution in distribution_list:
        filtered_distribution = [{k: v} for k, v in distribution.items() if v > theta]
        t1_list.extend(filtered_distribution)
    return t1_list
Iter['DTC'] = Iter['sentence_topic_distributions'].swifter.apply(filter_topic_distribution)

#coherence score
def com_coh(m sent, dict, cor):
    cohmodel_lda = CoherenceModel(model=m, texts=sent,
    dictionary=dict, coherence='c_v')
    coh = cohmodel_lda.get_coherence()
    return coh

def train(row):
    print(row.name, end='')
    tok_s = [word_tokenize(sent) for sent in row['processed']]
    dictionary = corpora.Dictionary(tok_s)
    cor = [dictionary.doc2bow(doc) for doc in tok_s]
    l-model = models.LdaModel(
        corpus=cor,
        id2word=dict,
        num_topics=5,
        random_state=42,
        alpha=0.5,
        eta=0.1,
        decay=0.5,
    )
    coh_s = com_coh(l-model, tok_s, dict, cor)
    sentence_topic_distributions = []

```

```

for doc in corpus:
    sentence_topic_distribution = lda_model.get_document_topics(doc)
    sentence_topic_distributions.append(dict(sentence_topic_distribution))
topic_words_list = []
for idx, topic in lda_model.print_topics(-1):
    topic_words = [word.split()[1] for word in topic.split()[1:] if "" in word]
    topic_words_list.append(topic_words)
row['topic_words'] = topic_words_list
row['sentence_topic_distributions'] = sentence_topic_distributions
row['coherence_score'] = coherence_score
return row

#evaluate summary
def evaluate_summary(df_summary):
    relevance_scores = []
    for summary, topic_keywords in zip(df_summary['Summary'], df_summary['top']):
        relevance_score = evaluate_relevance(summary, topic_keywords)
        relevance_scores.append(relevance_score)
    df_summary['Relevance Score'] = relevance_scores
    return df_summary

#cross fold validation
models = {
    "Linear SVC": LinearSVC(),
    "Random Forest": RandomForestClassifier(),
    "Ensemble (SVC, NB, LR, RF)": VotingClassifier(
        estimators=[
            ('svc', LinearSVC()),
            ('nb', MultinomialNB()),
            ('lr', LogisticRegression()),
            ('rf', RandomForestClassifier())
        ],
        voting='hard'
    ),
    "Ensemble (SVC, RF)": VotingClassifier(
        estimators=[
            ('svc', LinearSVC()),
            ('rf', RandomForestClassifier())
        ],
        voting='hard'
    )
}
folds = [2, 3]
mean accuracies = {}
for model_name, model in models.items():

```

```

for fold in folds:
    pipeline = Pipeline([
        ('tfidf', TfidfVectorizer()),
        ('classifier', model)
    ])
    cv = StratifiedKFold(n_splits=fold, shuffle=True, random_state=42)
    cv_scores = cross_val_score(pipeline, summary_df1['preprocessed_sentence'],
summary_df1['TBLLabel'], cv=cv)
    mean_accuracy = np.mean(cv_scores)
    mean_accuracies[(model_name, fold)] = mean_accuracy
fig, ax = plt.subplots(figsize=(12, 8))
x = np.arange(len(models) * len(folds))
labels = [f"{model_name} ({fold}-fold CV)" for model_name, fold in mean_accuracies.keys()]
heights = mean_accuracies.values()
ax.bar(x, heights, tick_label=labels)
ax.set_ylabel('Mean Accuracy')
ax.set_xticks(x)
ax.set_xticklabels(labels, rotation=45, ha='right')
ax.set_title('Mean Accuracy of Various Models with Different Cross-Validation Folds')
plt.tight_layout()
plt.show()
#reaching definition
def calculate_reaching_definition(df,r):
    definitions_generated = []
    definitions_killed = []
    for index, row in df.iterrows():
        row_definitions_generated = [set() for _ in range(len(row['processed']))]
        row_definitions_killed = [set() for _ in range(len(row['processed']))]
        for i, (sentence, topic_idx, sentiment_score) in enumerate(zip(row['processed'], row['ST'], row[r])):
            definition_generated = (topic_idx, sentiment_score)
            for j in range(i):
                if (row['ST'][j], row[r][j]) == definition_generated:
                    row_definitions_killed[i].add(j)
            row_definitions_generated[i].add(definition_generated)
        for j in range(i):
            if j not in row_definitions_killed[i]:
                row_definitions_generated[i] |= row_definitions_generated[j]
        definitions_generated.append(row_definitions_generated)
        definitions_killed.append(row_definitions_killed)
    df['definitions_generated'] = definitions_generated
    df['definitions_killed'] = definitions_killed
    return df
#average fluency and relevant score
num_dataframes = len(dataframes)
indi= np.arange(num_dataframes)
bwidth = 0.35
figu, axs = plt.subplots(figsize=(10, 6))
fluency_bars = axs.bar(indi - bwidth/2, f_scores, bwidth, label='Fluency Score', color='salmon')
relevance_bars = axs.bar(indi + bwidth/2, r_scores, bwidth, label='Relevance Score',
color='lightgreen')
axs.set_xlabel('LDA Topics')

```

```

axs.set_ylabel('Score')
axs.set_title('Average Fluency and Relevance Scores for Each Topic')
axs.set_xticks(indi)
axs.set_xticklabels([f"{i+5}" for i in range(num_dataframes)])
axs.legend()

def autolabel(bars):
    for b in bars:
        height = b.get_height()
        axs.annotate(' {:.2f}'.format(height),
                    xy=(b.get_x() + b.get_width() / 2, height),
                    xytext=(0, 3),
                    textcoords="offset points",
                    ha='center', va='bottom')
autolabel(f_bars)
autolabel(r_bars)
plt.tight_layout()
plt.show()

#pipeline
pi = make_pipeline(TfidfVectorizer(), LinearSVC())
cv = StratifiedKFold(n_splits=3, random_state=89)
cv_scores = cross_val_score(pipeline, summary_df['preprocessed_sentence'], summary_df['TBLLabel'],
                             cv=cv)
print("Cross-validation scores:", cv_scores)
print("Mean accuracy:", cv_scores.mean())

#random forest classifier
accuracy = []
pipeline = make_pipeline(TfidfVectorizer(), RandomForestClassifier())
cv = StratifiedKFold(n_splits=3, shuffle=True, random_state=42)
for df in dataframes:
    df['preprocessed_sentence'] = df['Summary'].apply(preprocess_text)
    cv_scores = cross_val_score(pipeline, df['preprocessed_sentence'], df['TBLLabel'], cv=cv)
    accuracy.append(cv_scores.mean())
crs = ['lightcoral', 'lightskyblue', 'mediumseagreen', 'gold', 'orchid', 'lightsteelblue']
plt.bar(range(1, len(dataframes) + 1), accuracy, color=crs)
plt.xlabel('Configurations')
plt.ylabel('Mean Accuracy')
plt.title('Mean Accuracy of 3 Fold Stratified Cross-validation for each Configuration
[RandomForestClassifier]')
plt.xticks(range(1, len(dataframes) + 1))
plt.show()

#linear svc
cv = StratifiedKFold(n_splits=3, random_state=99)
for df in dataframes:
    df['preprocessed_sentence'] = df['Summary'].apply(preprocess_text)
    cv_scores = cross_val_score(pipeline, df['preprocessed_sentence'], df['TBLLabel'], cv=cv)
    accuracy.append(cv_scores.mean())
colors = ['lightcoral', 'lightskyblue', 'mediumseagreen', 'gold', 'orchid', 'lightsteelblue']
plt.bar(range(1, len(dataframes) + 1), accuracy, color=colors)

```

#metrics

```
Iter12 = data.swifter.apply(train, axis=1)
Iter12['DTC'] = Iter12['sentence_topic_distributions'].swifter.apply(filter_topic_distribution)
Iter12['ST']=Iter12['DTC'].swifter.apply(get_dominant_topics)
Iter12['ATW'] = Iter12.swifter.apply(get_topic_words, axis=1)
Iter12['sentiment_score'] = Iter12.apply(lambda row: match_and_calculate_sentiment(row['processed'],
row['ATW']), axis=1)
df12=calculate_reaching_definition(Iter12,'sentiment_score')
IN_sets12, OUT_sets12 = calculate_IN_OUT_sets(df12['definitions_generated'],
df12['definitions_killed'])

S12 = [0] * len(OUT_sets12)
for i, out_set in enumerate(OUT_sets12):
    S12[i] = list(out_set)[-1] if out_set else None

summaries12, sentiment_scores_summary12 = generate_summary(Iter12['sentence'], S12,
Iter12['sentiment_score'])
summary_df12 = pd.DataFrame({'Summary': summaries12, 'Sentiment Score':
sentiment_scores_summary12})

Iter12['TBLLabel']=Iter12['Reviews'].swifter.apply( classify_sentiment)
summary_df12['TBLLabel']=summary_df12['Summary'].swifter.apply( classify_sentiment)

act = Iter12['TBLLabel']
pred = summary_df12['TBLLabel'] label
=['negative','neutral','positive']
cm = confusion_matrix(actual, predicted)
labels = np.unique(actual)

accuracy12 = accuracy_score(act, pred)
precision12 = precision_score(act, pred, average='weighted') recall12 =
recall_score(act, pred, average='weighted')
f112 = f1_score(act, pred, average='weighted')
print("Accuracy:", accuracy12)
print("Precision:", precision12)
print("Recall:", recall12)
print("F1-score:", f112)
```


REFERENCES

CHAPTER10

REFERENCES

10.1 REFERENCES

- [1] Li, X., Zou, C., Wu, P., & Li, Q. (2023). TSSRD: A Topic Sentiment Summarization Framework Based on Reaching Definition. "IEEE Transactions on Affective Computing", 14(3)
- [2] Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent Dirichlet allocation. "Journal of Machine Learning Research", 3, 993–1022.
- [3] Kalarani, P., & Selva Brunda, S. (2019). Sentiment analysis by POS and joint sentiment topic features using SVM and ANN. "Soft Computing", 23(16), 7067–7079.
- [4] M. Jang and P. Kang, "Learning-free unsupervised extractive summarization model" IEEE Access, vol. 9, pp. 14358–14368, 2021.
- [5] Elbarougy, R., Behery, G., & El Khatib, A. (2020). Extractive Arabic Text Summarization Using Modified PageRank Algorithm. "Egyptian Informatics Journal", 21(2), 73-81.
- [6] F. Huang, X. Li, C. Yuan, S. Zhang, J. Zhang, and S. Qiao, "Attention-emotion-enhanced convolutional LSTM for sentiment analysis," IEEE Transactions on Neural Networks.
- [7] Elgeldawi, E., Sayed, A., Galal, A. R., & Zaki, A. M. (2021). "Hyperparameter Tuning for Machine Learning Algorithms Used for Arabic Sentiment Analysis". Informatics, 8(4), 79. <https://doi.org/10.3390/informatics8040079>.
- [8] Gurusamy, V., & Kannan, S. (October 2014). "Preprocessing Techniques for Text Mining." Presented at RTRICS At: Podi. Madurai Kamaraj University. Available online: https://www.researchgate.net/publication/273127322_Preprocessing_Techniques_for_Text_Mining.
- [9] Masud, A. N., & Ciccozzi, F. (2020). "More precise construction of static single assignment programs using reaching definitions". Journal of Systems and Software, 166, Article no. 110590.
- [10] J.-Y. Wang and X.-L. Zhang, "Deep NMF Topic Modeling," Journal of LaTeX Class Files, vol. 14, no. 8, August 2015, pp. 1-1.
- [11] X. Wang, A. McCallum, and X. Wei, "Topical N-grams: Phrase and Topic Discovery, with an Application to Information Retrieval," University of Massachusetts.


- [12] Kharde, V. A., & Sonawane, S. S. (2016). "Sentiment Analysis of Twitter Data: A Survey of Techniques". *International Journal of Computer Applications*, 139(11), 1-6.
- [13] Lazrig, I., & Humpherys, S. L. (2022). "Using Machine Learning Sentiment Analysis to Evaluate Learning Impact". *Information Systems Education Journal (ISEDJ)*, 20(1), Article 13. ISSN: 1545-679X. <https://isedj.org/>.
- [14] W. Zhang, Y. Deng, B. Liu, S. J. Pan, and L. Bing, "Sentiment Analysis in the Era of Large Language Models" May24,2023. <https://arxiv.org/abs/2305.15005>.
- [15] Yadav, Arun Kumar, Singh, Amit, Dhiman, Mayank, Vineet, Kaundal, Rishabh, Verma, Ankit, & Yadav, Divakar. "Extractive text summarization using deep learning approach." *International Journal of Information Technology*, vol. 14, no. (2022): pp. 2407–2415. Published 19 January 2022.
- [16] Yadav, Avaneesh Kumar, Ranvijay, Yadav, Rama Shankar, & Maurya, Ashish Kumar. (2023). State-of-the-art approach to extractive text summarization: a comprehensive review. *Multimedia Tools and Applications*, 82, 29135–29197.
- [17] R. Nallapati, B. Zhou, C. Gulcehre, B. Xiang, "Abstractive Text Summarization Using Sequence-to-Sequence RNNs and Beyond," *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning (CoNLL)*, 2016.
- [18] K. Zhang, Y. Wu, "Fast Abstractive Summarization with Reinforce-Selected Sentence Rewriting," *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2017.
- [19] P. V. Kulkarni, R. Aljadda, B. J. P. H. Ribeiro, "RedSum: A Redundancy-Aware Summarization Dataset," *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2020.
- [20] A. See, P. J. Liu, C. D. Manning, "Get To The Point: Summarization with Pointer-Generator Networks," *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2017.

ANNEXURE

CHAPTER 11

ANNEXURE

11.1 JOURNAL SUBMISSION PROOF



INDERSCIENCE *Submissions*
Article submission and peer-review system

Help

Logged in as:
ajay2152002
[\[Log out\]](#)

[Inderscience Submissions Dashboard](#) | [Support & Documentation](#) | [Inderscience Home](#)

Dear **T. Ajay Kumar**,

Thank you! You have successfully submitted your article "*TSDA: TOPIC STABILITY DRIVEN APPROACH WITH DATA FLOW ANALYSIS AND HYPERPARAMETER OPTIMIZATION FOR ENHANCED TOPIC SENTIMENT SUMMARIZATION*" for the Int. J. of Computational Science and Engineering.

Your submission code is **JCSE-204407**

Please use this code if you contact us regarding your submission.

We will now check over your submission to verify that all is in order. If everything is ok, you will receive an automatic submission acknowledgement email and your article will proceed to the peer-review stage.

If your article does not pass our screening process, we will contact you directly.

It will take several months to review your article. If your article does not appear to be under review 5 months after submission, please contact the editor of the journal.

If you encounter any problems, you can contact us at submissions@inderscience.com

We thank you for submitting your article and for your interest in Int. J. of Computational Science and Engineering (JCSE).

[View Your Submissions](#)

[Contact us](#) | [About Inderscience](#) | [OAI Repository](#) | [Privacy and Cookies Statement](#) | [Terms and Conditions](#) | © Inderscience Enterprises Ltd. 2024

11.2 PAPER PLAGIARISM PROOF

Paper tsda

ORIGINALITY REPORT

1%

SIMILARITY INDEX

1%

INTERNET SOURCES

0%

PUBLICATIONS

0%

STUDENT PAPERS

MATCH ALL SOURCES (ONLY SELECTED SOURCE PRINTED)

< 1%

★ Anand Babu GI, Srinivasu Badugu. "Extractive Summarization of Telugu Text Using Modified Text Rank and Maximum Marginal Relevance", ACM Transactions on Asian and Low-Resource Language Information Processing, 2023

Publication

Paper tsda

by Ajay kumar

Submission date: 09-Apr-2024 10:34AM (UTC+0530)

Submission ID: 2344326909

File name: FInal_TSDA_Journal_1.pdf (1.12M)

Word count: 6114

Character count: 35030

11.3 REPORT PLAGIARISM PROOF

Report tsda

ORIGINALITY REPORT

7 %

SIMILARITY INDEX

5 %

INTERNET SOURCES

2 %

PUBLICATIONS

4 %

STUDENT PAPERS

MATCH ALL SOURCES (ONLY SELECTED SOURCE PRINTED)

2%

★ www.coursehero.com

Internet Source

Exclude quotes On

Exclude matches < 5 words

Exclude bibliography On