AN ACADEMIC META REVIEW GENERATION SYSTEM AND DECISION PREDICTION OF SCHOLARLY ARTICLE

A Dissertation submitted in partial fulfillment of the requirements for the degree

Of

Bachelor Of Technology

In

Computer Science and Engineering

Submitted by

MOUSTAFA SHAHIN (CSB19015)

HRISHIKESH GOGOI (CSB19030)

Under the supervision of

DR. TRIBIKRAM PRADHAN

Asst. Professor

School Of Engineering

Department of Computer Science and Engineering

Tezpur University

Napaam – 784028, Assam, India

June 2023

Department Of Computer Science and Engineering

Tezpur University

CERTIFICATE BY THE EXAMINER

This dissertation titled "An Academic Meta Review Generation System and Decision Prediction of Scholarly Article" submitted by Moustafa Shahin (CSB19015) and Hrishikesh Gogoi (CSB19030) in partial fulfilment of the requirements for the major project of Bachelor of Technology in Computer Science and Engineering has been examined.

Examiner

Date:

Place: Tezpur

Department Of Computer Science and Engineering

Tezpur University

Certificate by the HoD

This is to certify that the dissertation entitled "An Academic Meta Review Generation System and Decision Prediction of Scholarly Article" is submitted by Moustafa Shahin bearing Roll no: CSB19015 and Hrishikesh Gogoi bearing Roll no: CSB19030. They have completed their project work successfully as needed for partial fulfilment of the requirements and the regulations for the award of the degree of Bachelor of Technology in Computer Science & Engineering during the session 2019-2023 at Tezpur University. To the best of my knowledge, the matter embodied in the dissertation has not been submitted to any other university/institute for the award of any Degree or Diploma.

Date:

Place:

Head of the Department

Department of Computer Science and Engineering

Tezpur University

Department Of Computer Science and Engineering

Tezpur University

CERTIFICATE

This is to certify that the dissertation entitled "An Academic Meta Review Generation System and Decision Prediction of Scholarly Article" is submitted by Moustafa Shahin bearing Roll no: CSB19015 and Hrishikesh Gogoi bearing Roll no: CSB19030 is carried out by them under my supervision and guidance for partial fulfilment of the requirements and the regulations for the award of the degree of Bachelor of Technology in Computer Science & Engineering during the session 2019-2023 at Tezpur University. To the best of my knowledge, the matter embodied in the dissertation has not been submitted to any other university/institute for the award of any Degree or Diploma.

Date:

Place:

Dr. Tribikram Pradhan
Assistant Professor
Department of Computer Science & Engineering
Tezpur University

Department Of Computer Science and Engineering

Tezpur University

DECLARATION

We hereby declare that the dissertation work titled "An Academic Meta Review Generation System and Decision Prediction of Scholarly Article" submitted to the Department of Computer Science and Engineering, Tezpur University is prepared by us and was not submitted to any other institute for award of any other degree.

Date:

Place:

Moustafa Shahin (CSB19015)

Hrishikesh Gogoi (CSB19030)

Department of Computer Science & Engineering

Tezpur University

Acknowledgement

We would like extend our heartful gratitude to our project guide Dr. Tribikram Pradhan, Assistant Professor, Department of CSE, Tezpur University, for giving us the opportunity to work under him and providing me ample guidance and support through the course of project.

We would also like to thank Head of the department of computer science and Engineering department and all the faculty members of department of CSE, Tezpur University for the valuable guidance and co-operation throughout the project.

Our thanks and appreciations also go to all other people who have directly or indirectly helped me out with their abilities.

Moustafa Shahin

Hrishikesh Gogoi

Abstract

Peer review is an essential aspect of scientific communication. Before they are officially accepted or rejected for publication or funding, research articles and proposals are reviewed by multiple colleagues. As the number of research topics, academic venues (journals and/or conferences), researchers, and publications increases, it becomes increasingly difficult to manage the peer review process. Therefore, research is being conducted on the use of recommender systems to facilitate peer review, which is becoming an increasingly popular area of study. In this paper, we present a system for generating academic meta-reviews. (i) predicting the optimal algorithm for summarization (Task 1) and (ii) producing meta-reviews (Task 2). The system generates a meta-review based on the peer evaluations that are entered. Then, forecast the ultimate decision regarding the scholarly article (accept or reject). The proposed method combines extractive summarization and machine learning algorithms to produce the final meta-review. Our method prioritises providing a concise meta-review while maximising information coverage, coherence, readability, and redundancy.

List Of Figures

List Of Tables

Contents

CHAPTER 1

1.1 INTRODUCTION

The enormous growth in academic data in the modern era has made it difficult to extract insightful and efficient information. Academic entities such as researchers, publications, and venues, as well as academic relationships such as co-authorship and inter-citation, have expanded rapidly. For instance, the DBLP1 dataset, a collection of scientific publication records and their relationships within the collection, contains more than 9585 computer science conferences and more than 41523 journals. In addition to providing ample opportunities for researchers, this phenomenon has also spawned new challenges, particularly in the field of peer-review system management. The exponential growth of the volume and variety of data requires the development of more sophisticated tools to aid in the exploration of academic data, and a substantial amount of effort has been expended to encourage the development of various types of useful applications. Reference recommendation, scholar profiling, and academic impact estimation are examples of representative works. Utilising academic evaluations, peer review extraction, and recommendation has been regarded as a useful application. Peer review is an indispensable component of scientific communication. It is used not only to evaluate research papers and proposals but also to determine funding. Peer review is an essential component of the scientific method; it serves to ensure that research is accurate, reliable, and conducted ethically. Peer reviewers are subject matter specialists who have conducted extensive research, so their opinions can aid in ensuring that a paper adheres to the standards of its discipline. As you may be aware, reviews are frequently written by multiple authors; a given paper may have multiple reviewers. Occasionally, these evaluations will be merged into a single recommendation, but not always! Therefore, despite the fact that you are submitting your paper for review by multiple individuals (and, more likely, multiple journals), you should ensure that there are no inconsistencies between their recommendations. Text summarization is an indispensable instrument in today's society, where data is generated at a phenomenal rate. There are two categories of text summarization: extractive and abstractive. While extractive text summarization focuses on selecting key sentences from the original document(s), abstractive summarization focuses on generating new sentences based on the document's key points. The sentences in extractive summaries are generally of superior quality because they were written by real people. The objective of the task of generating a meta-review from the provided peer reviews is also to select the most important sentences or points from the peer review and present the pertinent information in a concise manner. However, the current text summarization methods cannot be directly

applied to the task of meta-review generation, as meta-reviews focus on specific aspects of a paper that can influence its acceptance or rejection. For instance, plagiarism in an article can result in immediate rejection, whereas misspellings typically have no bearing on the decision. In addition, existing summarization techniques do not differentiate between positive and negative points in a summary; consequently, they are unable to determine which points are strong or weak.

1.2 PROBLEM STATEMENT

Problem definition: Automatically predicting the final decision of peer reviews for scholarly papers is a tedious task due to heterogeneous nature of its entities. In this segment, we exhibit the problem description and elaborate on various notations and terminology.

Problem statement: Peer review is the method by which a group of scholars evaluates the work of another scholar. Peer reviews must be precise for them to be regarded as an acceptable method of evaluation. However, it is common for scholars to submit manuscripts with punctuation and grammar errors, making them difficult for peers to evaluate. In addition, the procedure is very time-consuming, given the importance of time in the research field. Therefore, an automated meta-review generation system is required.

Objectives:

- To conclude the best summarization algorithm present for academic review summarization for the task of meta review generation.
- To develop a software that will take human-generated reviews of a dataset and generate a machine-generated meta-review. It will compare all the existing summarization algorithms and techniques (both extractive and abstractive) and find the best method to use in order to develop an an algorithm that can generate meta reviews of a dataset automatically. And based on the generated algorithm it can predict the acceptance or rejection of a particular paper.

1.3 META REVIEW

Meta-analysis is a systematic review (Meta Review) of the literature that provides quantitative estimates of the effects of treatment interventions or exposures. A meta-analysis can be used to identify areas where there is uncertainty in the body of data on a topic and where there are relatively homogenous studies. Meta-analysis techniques provide standardized approaches for analysing prior

findings in a specific topic in the literature. Meta-analysis findings may not only be quantitative but also may be qualitative, revealing biases, strengths, and weaknesses in existing studies. The results of a meta-analysis can be used to form treatment recommendations or to provide guidance for future clinical trials.

Meta-analysis is a method of combining the results from multiple studies to determine the effect of a treatment or exposure. Meta-analysis provides a standardized approach for examining the existing literature on a specific, possibly controversial, issue to determine whether a conclusion can be reached regarding the effect of a treatment or exposure.

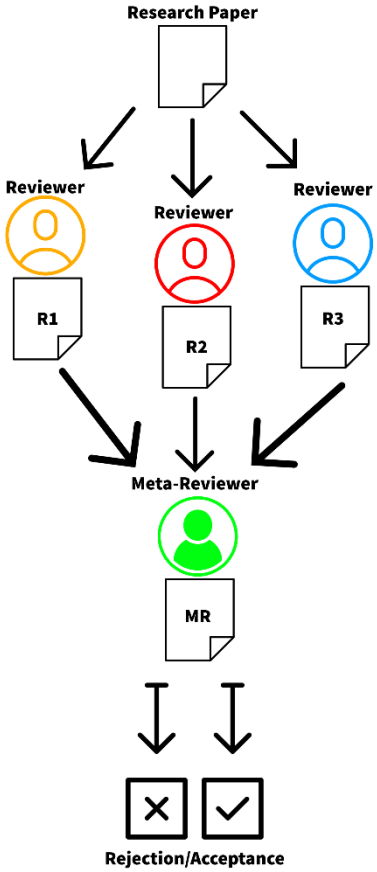


Fig 1.1: Meta-Review Generation system

Meta-analysis is often used when there are several small studies that could not be combined into one large study due to insufficient power. For example, 8 studies of streptokinase suggested its effectiveness in treating patients presenting with myocardial infarction, yet only 3 of these studies reported statistically significant results. Nevertheless, the results of a meta-analysis combining data

across all 8 studies concluded that streptokinase was associated with a statistically significant reduction in mortality.

The strength of meta-analysis lies in its ability to combine the results from various small studies that may have been underpowered to detect a statistically significant difference in effect of an intervention.

1.4 DEMO OF METAREVIEW

1.4.1 Reviews:

Review 1: This paper argues that text generated by existing neural language models are not as good as real text and proposes two metric functions to measure the distributional difference between real text and generated text. The proposed metrics are tried on language GANs but fail to produce any improvement. Major issues: This manuscript is poorly organized and the introduction is not well-written. It's true that generating text from random noise vector remains a challenging problem, but sequence-to-sequence models for machine translation and question answering have achieved tremendous successes. The description in the first paragraph about neural language models is not accurate. There are numerous grammar issues and mis-spellings. For e.g., pp. 1: "RelGAN which needs not...", pp. 2: "We analysis…", "could be find…", pp 3: "equation 8" should be "equation 9"... The proposed metrics are also questionable. Eq. 3 on page 2 holds for any x sampled from the distribution, not just for a single data point. To test the effectiveness of a good metric, extensive experiments on toy datasets such as MNIST, CIFAR10, and synthetic datasets should be conducted. This paper mixes text generation and proposed metrics together. The claimed failure experiments make the proposed metrics even more questionable. In summary, the presentation and the organization of this paper should be significantly improved for submission. The proposed metrics are questionable and should be thoroughly tested on synthetic and toy datasets before deploying it for text generation.

Review 2: This paper proposes two metrics to measure the discrepancy between generated text and real text, based on the discriminator score in GANs. Empirically, it shows that text generated by current text generation methods is still far from human-generated text, as measured by the proposed metric. The writing is a bit rough so sometimes it's hard to figure out what has been done. It's also unclear how the proposed metrics compare to simply using the discriminator for evaluation. Therefore, I'm inclined to reject the current submission. Approach: - The proposed metric essentially relies on the learned discriminator to measure the closeness of generated text vs real text, based on the strong assumption that the learned discriminator is near-optimal. It has been previously shown that learning a classifier from generated and real text does not generalize well (Lowe et al, 2017, Chaganty et al, 2018). - What's the advantage of the proposed metric, compared to existing ones, e.g. KL divergence, total variation etc.? Experiments: - What's the accuracy of the learned discriminators? The discrepancy could be due to both data difference and classification error. Minor: Bleu -> BLEU Reference: Towards an automatic turing test: Learning to evaluate dialogue responses.

R. Lowe, M. Noseworthy, I. V. Serban, N. Angelard- Gontier, Y. Bengio, and J. Pineau. 2017. The price of debiasing automatic metrics in natural language evaluation. A. Chaganty, S. Mussmann, and P. Liang. 2018.

Review 3: Pros:

This paper argues that text generated by existing neural language models are not as good as real text and proposes two metric functions to measure the distributional difference between real text and generated text.

However, the method to achieve this is quite convoluted, and straightforward generator training to minimize D_phi does not appear to work (the authors do not say why either). The authors are proposing a measure of discrepancy, which is essentially useful as a two-sample statistical test. As such, the authors should demonstrate a power analysis of their test to detect differences between real vs generated text and show this new test is better than tests based on existing discrepancy measures. In summary, the presentation and the organization of this paper should be significantly improved for submission. Overall, the writing also contains many grammatical errors and confusing at places. There are tons of other existing measures of distributional discrepancy that could be applied to this same problem. The price of debiasing automatic metrics in natural language evaluation. The description in the first paragraph about neural language models is not accurate. x needs to be defined before equation (1). It is mathematically incorrect to talk about probability density functions when dealing with discrete text. Rather these should be referred to as probability mass functions, likelihoods, or distributions (not "distributional function" either). To test the effectiveness of a good metric, extensive experiments on toy datasets such as MNIST, CIFAR10, and synthetic datasets should be conducted. Reference: Towards an automatic turing test: Learning to evaluate dialogue responses.

Cons: Kullback-Leibler or other f-divergence based on estimated densities, Maximum Mean Discrepancy based on a specific text kernel, etc) while others would be highly related to this work through their use of a classifier. The writing is a bit rough so sometimes it's hard to figure out what has been done. It's also unclear how the proposed metrics compare to simply using the discriminator for evaluation. Therefore, I'm inclined to reject the current submission. The proposed metrics are tried on language GANs but fail to produce any improvement. The discrepancy could be due to both data difference and classification error. The proposed metrics are questionable and should be thoroughly tested on synthetic and toy datasets before deploying it for text generation. The claimed failure experiments make the proposed metrics even more questionable. This manuscript is poorly organized and the introduction is not well-written. It’s true that generating text from random noise vector remains a challenging problem, but sequence-to-sequence models for machine translation and question answering have achieved tremendous successes. Given all these existing methods (I am sure there are many more), it is unclear to me why the estimator proposed in this paper should be better. The authors need to clarify this both intuitively and empirically via comparison experiments (theoretical comparisons would be nice to see as well).

Eq 3 on page 2 holds for any x sampled from the distribution, not just for a single data point.

1.4.2 Human Generated Meta Review:

The authors propose a novel metric to detect distributional discrepancy for text generation models and argue that these can be used to explain the failure of GANs for language generation tasks. The reviewers found significant deficiencies with the paper, including: 1) Numerous grammatical errors and typos, that make it difficult to read the paper. 2) Mischarcterization of prior work on neural language models, and failure to compare with standard distributional discrepancy measures studied in prior work (KL, total variation, Wasserstein etc.). Further, the necessity of the complicated procedure derived by the authors is not well-justified. 3) Failure to run experiments on standard banchmarks for image generation (which are much better studied applications of GANs) and confirm the superiority of the proposed metrics relative to standard baselines. The reviewers were agreed on the rejection decision and the authors did not participate in the rebuttal phase. I therefore recommend rejection.

1.4.3 Acceptance / Rejection Status: Rejected.

1.5 TEXT SUMMARIZATION

In text summarization, a method of reducing lengthy documents into manageable paragraphs or sentences, the procedure extracts key information while preserving the paragraph's meaning. This reduces the time required to comprehend lengthy materials, such as research articles, while ensuring that all essential details are included. Text summarising is the process of creating a concise, coherent, and fluid summary of a longer text document. Text summarization presents a number of challenges, including text identification, interpretation, summary generation, and summary analysis. In extraction-based summarising, identifying key phrases and exploiting them to find relevant information to include in the summary are crucial tasks. The quantity of text data available from various sources has multiplied in the era of big data. This extensive text contains a multitude of knowledge and information that must be adequately summarised for it to be useful. The increasing availability of documents necessitates extensive research in the field of natural language processing (NLP) for automatic text summarization. Automatic text summarization is the process of creating a concise and fluent summary without the assistance of a human while maintaining the original text's meaning. It is challenging because, in order to summarise a piece of literature, we typically read it in its entirety to gain a better understanding of it and then write a summary emphasising its key points. Because computers lack human language and comprehension, the automated summarization of text is a difficult and time-consuming process. In addition, text summarization reduces reading time, accelerates the research process, and increases the amount of information that can fit into a given space.

Based on Outcome There are mainly two types of text summarization in NLP:

1. Extractive summarization

The method of extractive text summarization involves taking particular words from a source and combining them to generate a summary. The extraction is carried out according to the specified standard without modifying the documents. The diagram below will assist in understanding the meaning of extractive summarization. This method identifies the essential portions of a text, eliminates them, and reassembles them to create a condensed version. They therefore rely solely on phrase extraction from the source text.

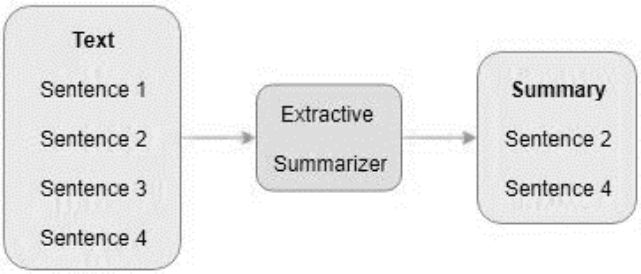


Fig 1.2: Extractive Summarization

2. Abstractive Summarization

Abstractive summarization is an approach that uses machine learning to generate new sentences based on a text's original content. This is in contrast to our previous extraction method, in which we only used the present phrases. It is possible that the phrases resulting from abstractive summarization do not appear in the original text. When used for text summarization in deep learning problems, abstraction can overcome the grammatical blunders of the extractive method. Abstract synthesis is more effective than extraction. In contrast, the text summarization algorithms required for abstraction are more difficult to construct, which is why extraction is still extensively used.

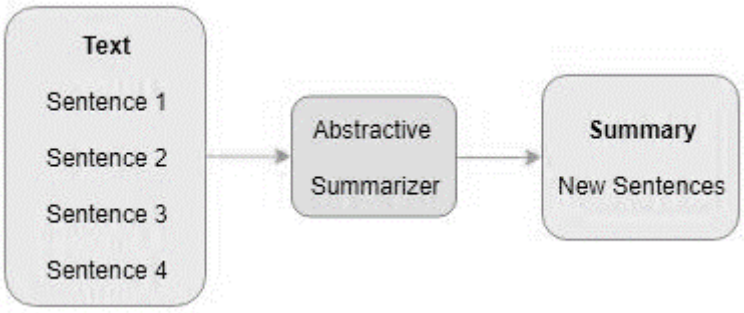


Fig 1.3: Abstractive Summarization

1.6 EXTRACTIVE SUMMARIZATION TECHNIQUES

1.6.1 LexRank Model

LexRank is an unsupervised method for the automated summarization of text based on graphs. Sentences are evaluated using a graphical technique. LexRank is used to determine the significance of a sentence based on the concept of eigenvector centrality within a graph representation of sentences. The adjacency matrix of the graph representation of sentences in this paradigm is a connectivity matrix based on intra-sentence cosine similarity. This sentence extraction concentrates on the set of sentences with the same function, i.e., a centroid sentence is selected to serve as the mean for all other sentences in the document. The similarities between the sentences are then assessed. This method does not require any pre-processing steps prior to commencing the summarization process, such as tokenization and stop word removal. This method can be implemented on any text processing system, such as C++, Python, or even Java, without the need to write code for data preparation or processing, as it employs only pre-trained word vector models for training.

Graphical Approach

1. Based on Eigen Vector Centrality.
2. Sentences are placed at the vertexes of the Graphs
3. The weight on the Edges are calculated using cosine similarity metric.

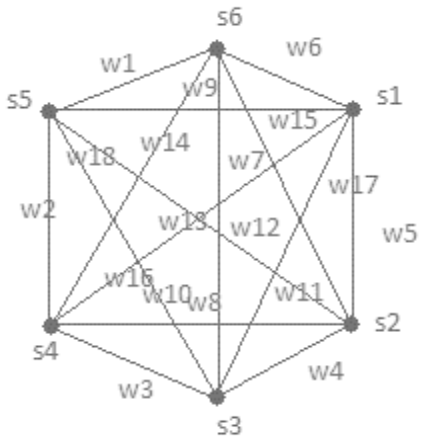


Fig 1.4: Lexrank Graphical model

Where S_i are the sentences at the vertices respectively and W_{ij} are weights on the edges.

1.6.2 TextRank Model

Based on Google's PageRank algorithm, TextRank is a graph-based ranking model for text processing. In 1998, Google's search engine used PageRank as its first algorithm for sorting web pages. If page A links to pages B and C, and page B links to page C, then the pages would be sorted as follows: page C, page B, and page A. Because TextRank is unsupervised, it is very simple to use. First, the algorithm divides the entire text into sentences, and then it constructs a graph in which the sentences are the nodes and the overlapping words are the connections. PageRank ultimately determines the most significant nodes in this network of sentences.

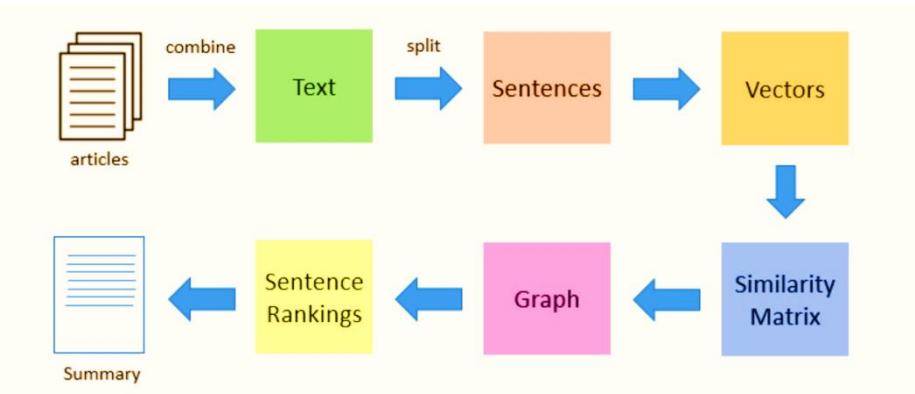


Fig 1.5: TextRank Model

- First, we must concatenate the entire article's text. This is achieved by obtaining the entire article text as a string and appending a space to it.

- The text is then divided into separate sentences. This can be achieved by isolating each sentence into its own string and then concatenating them.
- ➢ Next, vector representations (word embeddings) are determined for each sentence. Using the Word2Vec architecture, the word embeddings are calculated so that each word is represented by the words that surround it
- ➢ Similarities between sentence vectors are then calculated and stored in a matrix.
- ➢ The similarity matrix is then converted into a graph, with sentences representing the vertices and similarity scores representing the edges, in order to calculate sentence rank. Each edge connects two sentences with similar similarity scores; these edges, which form directed paths or arcs, are referred to as paths or arcs.
- Finally, a certain number of top-ranked sentences form the final summary

1.6.3 BERT Model

BERT (bidirectional transformer) is a transformer used to overcome the long-term dependencies of RNNs and other neural networks. It is an inherently bidirectional model that has already been trained. This pre-trained model can be readily tuned to execute the specified NLP tasks, in our case, summarization.

Till the date, BERTS are considered as the best available technique to perform the NLP tasks.

Points to a glance:

- It uses a powerful flat architecture with inter sentence transform layers so as to get the best results in summarization.
- BERT models are pre-trained on huge datasets thus no further training is required.
- The Summary sentences are assumed to be representing the most important points of a document.

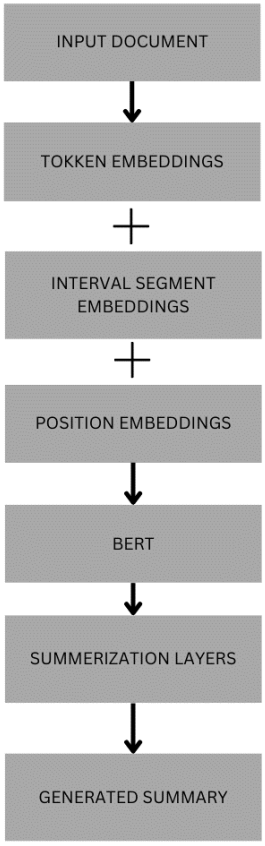


Fig 1.6: BERT Model

1.7 MACHINE LEARNING AND DEEP LEARNING CLASSIFICATION MODELS

1.7.1 Logistic Regression:

Logistic regression is a statistical technique for modelling the association between a binary dependent variable and one or more independent variables. It employs the logistic function to estimate the probability of a binary outcome, allowing for the classification of new observations according to their input characteristics.

1.7.2 DNN Model:

Deep neural networks (DNN) are artificial neural networks consisting of multiple layers of interconnected nodes, each of which implements a nonlinear activation function on the weighted sum

of its inputs. The layers are arranged progressively, with the output of each layer serving as the input for the following layer. DNNs can learn and extract hierarchical data representations autonomously, allowing them to model complex patterns and relationships.

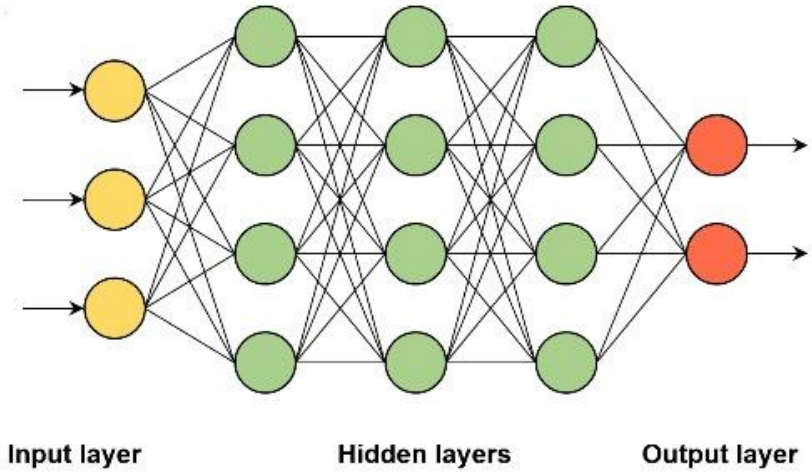


Fig 1.7: structure of DNN model

1.7.3 CNN Model:

Logistic regression is a statistical technique used to model the relationship between a binary dependent variable and one or more independent variables. It uses the logistic function to estimate the probability of a binary outcome, enabling the classification of new observations based on their input characteristics.

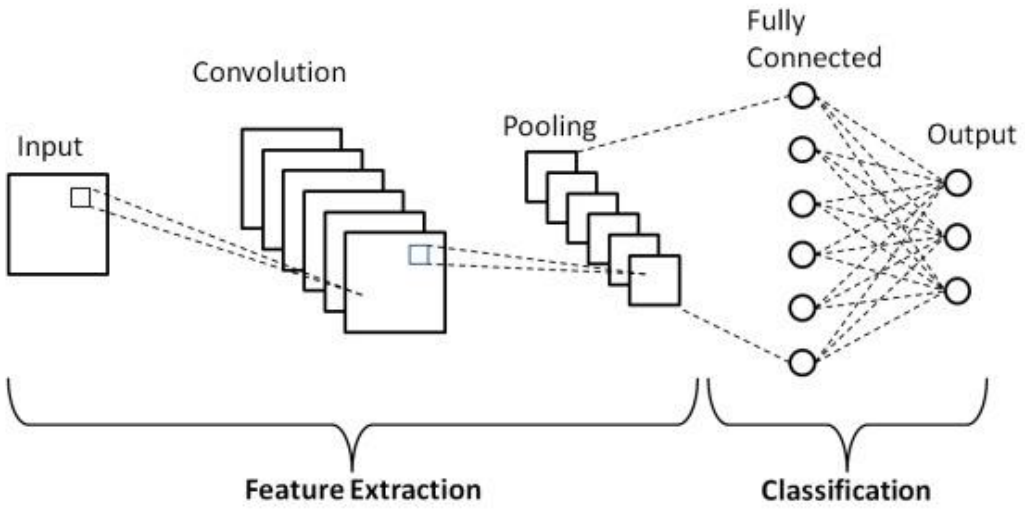


Fig 1.8: Schematic diagram of a basic convolutional neural network (CNN) architecture

1.7.4 LSTM Model

Long Short-Term Memory (LSTM) is an architecture for recurrent neural networks (RNNs) designed to overcome the limitations of conventional RNNs in capturing and storing long-term dependencies in sequential data. It excels at handling and analysing sequential data, including time series, text, and speech, among others.

Formally, LSTM is a subtype of RNNs that employs gated recurrent units to selectively learn and discard information across multiple time steps. It includes a memory cell state and three types of gates: an input gate, a neglect gate, and an output gate. These gates control the flow of information and regulate the cell's state update and utilisation.

The input gate determines how much of the new input is to be incorporated into the cell state. The forget gate determines how much of the preceding cell state is to be disregarded. This gating mechanism enables LSTMs to retain or discard information selectively over protracted sequences, making them ideal for capturing dependencies spanning multiple time steps.

1.8 METRICS

Abstractive summarization is typically achieved by paraphrasing text passages, whereas extractive summarization uses oracle summaries. The difference between these two approaches lies in the manner in which they manage data. The primary difference between the two approaches is that the first relies on human-generated summaries to generate a prediction, whereas the second uses computer-generated summaries as an oracle. Typically, humans abstract the golden annotated abstractive summaries. As a result, supervised learning is superfluous due to the well-defined objective. For sentence-wise Extractive summarization, however, an additional step is sometimes taken: a subset of sentences (with the highest similarity) from the original text passage is extracted based on golden annotated abstractive summaries and their number of sentences. ROUGE score is the metric used to quantify the similarity between two entities.

1.8.1 ROUGE Score

ROUGE scores are used to evaluate the closeness of model-generated summaries with golden annotated summaries.

The paper ROUGE: A Package for Automatic Evaluation of Summaries defines Recall-Oriented Understudy for Getting Evaluation (ROUGE). As its name suggests, ROUGE is "recalled-oriented,' but it also takes recall and precision into consideration when comparing candidate (model-generated or predicted) and reference (golden-annotated or target) summaries. In addition, ROUGE scores are categorised as ROUGE-1, ROUGE-2, and ROUGE-L.

Precision and Recall in the Context of ROUGE

Recall, in the context of ROUGE simply means how much of the reference summary is the system summary recovering or capturing?

If we only consider individual words, then recall can be determined by counting the number of times each word appears in a document. If we examine a document and count all occurrences of a particular word, we are performing a "recall.

“Therefore, if 80% of the words in a 100-word document have been used in other documents with similar content, we would say that our system has 80% recall.

It can be calculated as,

$$\frac{number\_of\_overlapping\_words}{total\_words\_in\_reference\_summary}$$

An important metric for computer-generated summaries (system summaries) is precision. We want to ascertain how much of the system summary was relevant or necessary. In other words, we desire to determine whether the summary accurately reflects your data.

A machine-generated summary (system summary) may be extremely extensive if it includes every word from the reference summary. However, many of the words in the system summary may be superfluous, resulting in an excessively wordy summary. This situation necessitates precision. Regarding precision, we are essentially assessing how much of the system summary was relevant or necessary.

It can be calculated as,

$$\frac{number\_of\_overlapping\_words}{total\_words\_in\_system\_summary}$$

1.8.1.1 ROUGE 1 score

Rouge-1 Compare is an innovative utility for comparing two summaries side-by-side. It allows you to compare the similarity of uni-grams between the candidate summary and the reference summary, which can assist you in deciding whether or not to alter your candidate summary for use in the real world.

By uni-grams, we mean that each token of comparison consists of a single word.

By Recall, we merely refer to the proportion of the reference summary's terms that were captured by the candidate summary.

Precision refers to the percentage of terms suggested by the candidate summary that are present in the reference summary.

1.8.1.2 ROUGE 2 score

Rouge-2 Precision and Recall is the tool that you need to compare the similarity of bi-grams between reference and candidate summaries. By bi-grams, we mean each token of comparison is 2 consecutive words from the reference and candidate summaries.

The similarity score between these two sets of bi-grams will be as high as possible. The higher the score, the more similar these bi-grams are to each other.

1.8.1.3 ROUGE L score

ROUGE-L Precision and Recall measures the longest common sequence (LCS) between the candidate and reference summaries. By LCS, we refer to sequential, but not necessarily consecutive, word tokens.By providing such a metric, we can identify data patterns that are not readily discernible from the summary text alone.

$$F1\ score = 2 * \frac{Precision\ * Recall}{Precision + Recall}$$

ROUGE-1, ROUGE-2, and ROUGE-L are excellent representations of how well the model-generated summaries depict the golden-annotated summaries. To make scores more concise, typically 22 F1 scores, the harmonic mean of precision and recall, are calculated for each ROUGE score.

1.8.2 Readability Score

Readability is the simplicity with which a reader can comprehend a piece of writing. Readability is frequently regarded as one of the most essential aspects of text production and analysis, especially when considering the effect of various reading strategies on comprehension. Word length, the proportion of multisyllabic words, and the presence of subordinate clauses in a text are the primary determinants of legibility (Muhlen-Bock & Kokkinakis, 2009).

Since the 1920s, readability has been the subject of study. Early research concentrated on vocabulary elements such as word length, the proportion of multisyllabic words, subordinate clauses, etc. Additionally, readability can be viewed from various angles. Partially, readability is the extent to which a reader can comprehend a written text; this has been studied for a long time because it has been beneficial to draw statistical conclusions on readability from experiments. This has resulted in the development of a number of formulations for various languages that make automated text measurement possible for longer texts.

1.8.2.1 Flesch–Kincaid Readability Tests

The Flesch-Kincaid readability tests are designed to indicate how difficult it is to comprehend an English passage. Flesch Reading Ease and Flesch-Kincaid Grade Level are the two available evaluations.

Although they use the same fundamental metrics (word length and sentence length), their weighting factors differ. The correlation between the two tests is approximately inverse: a text with a relatively high reading ease test score should have a reduced grade-level test score. Rudolf Flesch created the Reading Ease assessment, but because it was created at a time when testing computers were scarce, he used an optical scanner instead of computer software to collect the data.

1.8.2.2 Flesch Reading-Ease

The Flesch-Kincaid intelligibility index is a straightforward method for determining a text's level of difficulty. It counts the number of words in each sentence and identifies those that are too long or too brief, as well as those that are difficult to comprehend due to verbosity, redundancy, or other factors.

The Flesch reading ease test is a quick and easy way to check your understanding of the material you're reading. The higher your score, the easier it will be for you to understand what you're reading. The formula for the Flesch reading-ease score (FRES) test is:

$$206.835 - 1.015\left(\frac{\text{total words}}{\text{total sentences}}\right) - 84.6\left(\frac{\text{total syllables}}{\text{total words}}\right)$$

Scores can be interpreted as shown in the table below.

Score	Difficulty
100.00–90.00	Very easy to read. Easily understood by an average 11-year-old student.
90.0–80.0	Easy to read. Conversational English for consumers.
80.0–70.0	Fairly easy to read.
70.0–60.0	Plain English. Easily understood by 13- to 15-year-old students.
60.0–50.0	Fairly difficult to read.
50.0–30.0	Difficult to read.
30.0–10.0	Very difficult to read. Best understood by university graduates.
10.0–0.0	Extremely difficult to read. Best understood by university graduates.

Table 1.1: Flesch Reading-Ease

1.8.2.3 Flesch–Kincaid grade level

The field of education makes extensive use of these readability tests. By presenting a score as a U.S. grade level, the "Flesch-Kincaid Grade Level Formula" makes it easier for teachers, parents, librarians, and others to assess the readability level of diverse books and texts. It can also represent the number of years of schooling normally required to comprehend this book if the calculation yields a value greater than 10. The formula for the Flesch–Kincaid grade level score test is:

$$0.39\left(\frac{\text{total words}}{\text{total sentences}}\right)+11.8\left(\frac{\text{total syllables}}{\text{total words}}\right)-15.59$$

Scores can be interpreted as shown in the table below

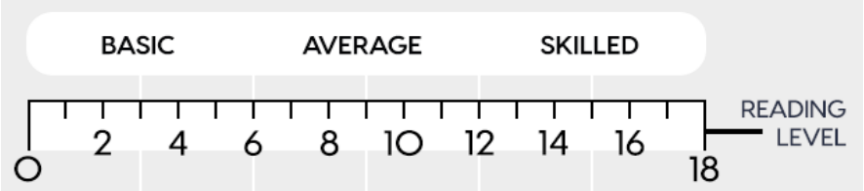


Fig 1.9: Grade levels

1.8.2.4 Automatic Readability Index

The automated readability index (ARI) is a readability test designed to determine the level of text comprehension in English. Similar to the Flesch-Kincaid grade level, the Gunning fog index, the SMOG index, the Fry readability formula, and the Coleman-Liau index, it approximates the grade level required to comprehend the text in the US.

The ARI considers numerous factors, including word count, average syllable length, and sentence length. The greater the number of variables included in your calculation, the more accurately you will represent a particular grade level. The formula for the Automatic Readability Index is:

$$4.71\left(\frac{\text{characters}}{\text{words}}\right)+0.5\left(\frac{\text{words}}{\text{sentences}}\right)-21.43$$

Scores can be interpreted as shown in the table below

Score	Age	Grade Level
1	5-6	Kindergarten
2	6-7	First Grade
3	7-8	Second Grade
4	8-9	Third Grade
5	9-10	Fourth Grade
6	10-11	Fifth Grade
7	11-12	Sixth Grade
8	12-13	Seventh Grade
9	13-14	Eighth Grade
10	14-15	Ninth Grade
11	15-16	Tenth Grade
12	16-17	Eleventh Grade
13	17-18	Twelfth Grade
14	18-22	College student

Table 1.2: Automatic Readability Index

1.8.3 Accuracy Score:

The accuracy of a machine learning or deep learning model for binary classification is the proportion of correctly predicted instances (true positives and true negatives) relative to the total number of instances evaluated. It is frequently used as a performance metric to assess the overall precision of the model's predictions. Here is the official definition of accuracy for binary classification models:

The formula for precision is (TP + TN) / (TP + TN + FP + FN).

Where:

TP (true positives): The number of positive instances that are correctly predicted as positive.

TN (true negatives): The number of negative instances that are correctly predicted as negative.

FP (False Positives): The number of negative instances that are incorrectly predicted as positive.

FN (False Negatives): The number of positive instances that are incorrectly predicted as negative.

CHAPTER 2

ALGORITHMS USED

2.1 LexRank Algorithm

The LexRank algorithm is divided into several processes:

1. The Input to the model.
   A single article or collection of articles could be used as the model's primary input. Whether you want to summarise one article or several articles will depend on your objective.

2. Word embeddings.
   To apply any complex algorithm, we must convert the textbook input into numbers, typically real-valued vectors. The representation of words in vector format is referred to as word embeddings. In general, word embeddings are deemed similar in the sense that words represented by similar vectors are anticipated to have similar meanings.

3. Intra-sentence cosine similarity.
   Word embeddings can be used in a sentence in a variety of ways. Still, in LexRank's implementation, a sentence's intra-sentence is utilised. It refers to the average of all word embeddings used to compare sentences. Cosine similarity is merely a metric for determining how similar data objects are, regardless of their sizes. In this instance, vectors represent similar sentences.
   The formula for calculating the cosine similarity is:
   Cos(x, y) = x . y / ||x|| * ||y||
   Where:
   x . y - a dot product
   || x || - length of a vector x
   || y || - length of a vector y

4. Adjacency matrix.

   LexRank employs a connectivity matrix to depict similarities between each sentence.

   When you have multiple papers, it is possible that a single paper will contain multiple similar sentences. In such a situation, the model could only consist of sentences from this single document. In additional specialised terms, this is known as a local snare, in which numerous insignificant sentences vote for each other. To alleviate the aforementioned issue, LexRank considers that the significance of a sentence should also derive from the significance of the sentences "recommending" it.

   Typically, an adjacency matrix is a binary matrix that only indicates the existence of an edge connecting two vertices.

5. Connectivity matrix.

   LexRank incorporates a count of connections from other sentences to eradicate the issue of local traps. There may be only two connections in a sentence, but they may be extremely potent and persuasive.

   LexRank employs a threshold to measure the number of connections. For instance, it only considers sentences similar to itself if the cosine similarity is greater than 0.3 or 0.1.

   A connectivity matrix is typically a list of the vertex numbers that have an edge between them.

6. Eigenvector centrality.

   LexRank makes use of eigenvector centrality to determine the most significant sentences. Power iteration method is the name of the technique.

7. Output of the model.

   The desired output is obtained.

2.2 BERT Algorithm

BERT (bidirectional transformer) is a transformer used to surmount the limitations of RNNs and other neural networks due to their long-term dependencies. It is an inherently bidirectional model that has already been trained. This pre-trained model can be modified to perform the specified NLP tasks, in our case, summarization, with ease.

BERT models have already been trained on enormous datasets, so no further training is required.

For optimal summarization results, it employs a robust flat architecture with multiple layers of sentence transformation.

Advantages:

1. It is most efficient summarizer till date.
2. It is faster than RNN.

Methodology:

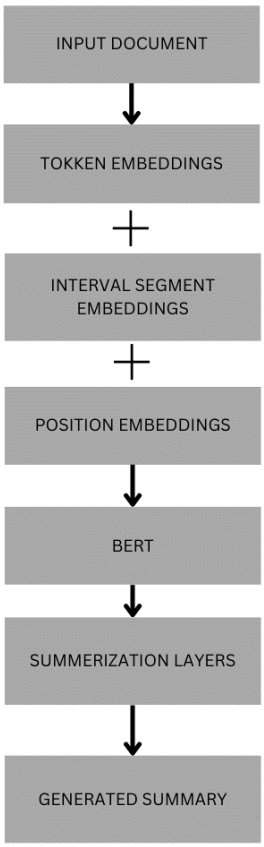


Fig 2.1: Methodology of BERT

Since the model was trained as a masked model, the output vectors are tokenized rather than sentences. In contrast to other extractive summarizers, this one employs embeddings to differentiate between sentences and has only two labels, A and B, as opposed to multiple sentences. Using modified versions of these embeddings, the necessary summaries are created.

The complete process can be divided into different phases:

1. Encoding Multiple Sentences:
   Sentences from the input document are encoded in this step in order to be pre-processed. Each sentence has a CLS tag before it and a SEP tag after it. The features of one or more sentences can be combined using the CLS tag.
2. Interval Segment Embeddings:
   Sentences in a document are distinguished in this step.

Embeddings:

It is primarily concerned with the representation of words as vectors. It makes their application more flexible. This BERT feature is used by Google to improve query comprehension. It helps to enable a variety of semantic functionality, such as determining the document's intent and constructing a model of word similarity.

There are three types of embeddings applied to our text prior to feeding it to the BERT layer:

1. Token Embeddings:
   Words are transformed into a vector with a fixed dimension. At the beginning and end of each sentence, respectively, [CLS] and [SEP] are added.
2. Segment Embeddings:
   It is used to separate, or we can say classify, various inputs using binary coding.
3. Position Embeddings:
   512-bit input sequences are supported by BERT. As a result, the vector's final dimensions will be (512,768). Due to the possibility that a word's placement in a sentence could change its context-specific meaning, positional embedding is used instead of vector representation.

BERT Architecture:

Two bert models introduced:

1. BERT base:
   We have 12 transformer layers, 12 attention layers, and 110 million parameters in the BERT base model.
2. BERT Large:

We have 24 transformer layers, 16 attention layers, and 340 million parameters in the BERT large model.

Transformer layer:

The transformer layer consists of an entire set of encoder and decoder layers as well as the intermediate connections. Each encoder consists of an RNN and an attention layer. The decoder has the same architecture as the seq2seq model, with the addition of an additional attention layer. It is advantageous to focus on essential terms.

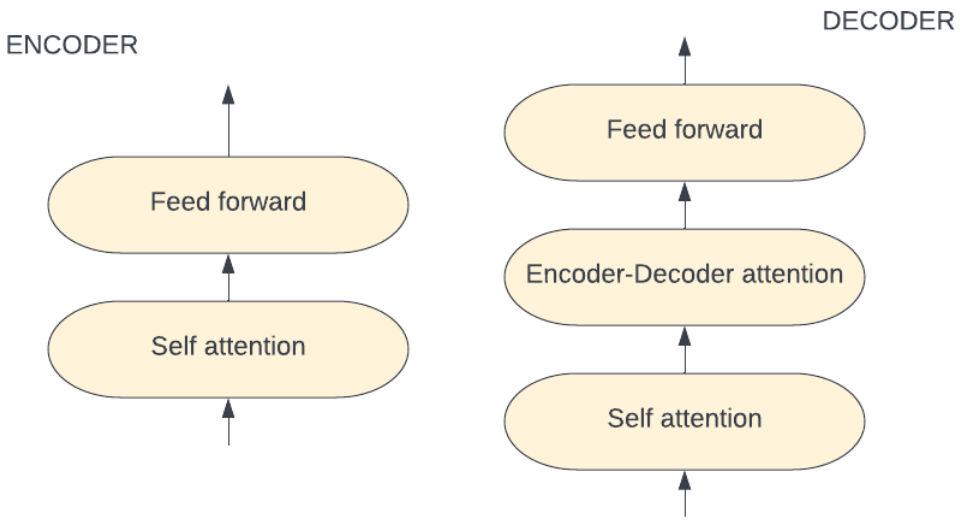


Fig 2.2: BERT Architecture

Summarization layers:

The Self attention layer is the only significant distinction between RNN and BERT. The model helps in representation by attempting to determine the strongest connections between the words.

Within the BERT model, we can have various kinds of layers, each with its own specifications:

1. Simple Classifier:
   To predict the score Yˆi, a linear layer and sigmoid function are added to the BERT in a straightforward classifier method.

2. Inter Sentence Transformer:
   In the intermediate sentence transformer, a simple classifier is not utilised. Instead, various transformer layers are only added to the model's sentence representation, enhancing its efficacy. This assists n i in identifying the important elements of the document.

3. Recurrent Neural network:
   The output of the BERT model is combined with an LSTM layer in order to learn the features specific to summarization. which normalises each LSTM cell.

2.3 TextRank Algorithm

Natural Language Processing uses TextRank, a text summarization algorithm, to produce document summaries.

TextRank uses an extractive approach and is an unsupervised graph-based text summarization technique.

Algorithm:

1. The quantity and calibre of links pointing to a particular page determine its rank and value.
2. To reach a final value, it does multiple iterations on the pages.

The following are the basic processes that make up the TextRank algorithm:

1. Remove every sentence from the text by partitioning it at whitespace, full stops, or wherever else you deem appropriate.
2. The sentences retrieved in Step 1 are used to create a graph.A similarity function, such as cosine similarity or Jaccard similarity, is used to determine the weight of the edges between two nodes.
3. In this phase, the algorithm is iterated until convergence, that is, until consistent scores are obtained, in order to determine the importance (scores) of each node.
4. The sentences are arranged in decreasing order by their scores. The summary of the text will include the first 10 sentences.

2.4 Logistic Regression Algorithm:

In logistic regression, the objective is to identify the optimal parameters that define a linear decision boundary separating the two classes. To accomplish this, the algorithm employs a method known as maximum likelihood estimation.

Here is a detailed explanation of how logistic regression operates:

Data Preparation: Preparing the data requires a labelled dataset in which each instance is associated with a class label. The input characteristics (or independent variables) must be numeric or converted to numeric values.

Model initialization: Initialise the weights (coefficients) and the bias term (intercept) to arbitrary values. During the training procedure, these weights and biases will be adjusted as parameters.

Training: The training procedure seeks to identify the optimal model parameter values. It begins by applying the logistic function to the input data to derive a probability estimate between 0 and 1. The logistic function consists of a linear combination of the input features, weighted by the learned coefficients, and the bias term.

$z = w1 * x1 + w2 * x2 +... + wn * xn + b$

where w1, w2,..., wn represent weights, x1, x2,..., xn represent input features, and b represents the bias term.

A threshold (typically 0.5) is applied to the output of the logistic function to determine the predicted class designation.

A loss function is used to quantify the difference between the predicted probabilities and the actual class designations. The most prevalent loss function in logistic regression is the binary cross-entropy loss (also known as log loss). The objective is to reduce this depletion during training.

Gradient Descent: The parameters (weights and bias) are modified iteratively using the Gradient Descent optimisation algorithm. The loss is minimised by calculating the gradient of the loss function with respect to the parameters and then adjusting the parameters in the opposite direction

of the gradient. This procedure is repeated a predetermined number of times or until convergence is reached.

Regularization (optional): Regularization techniques such as L1 or L2 regularisation can be used to prevent overfitting and enhance generalisation. They include a penalty term in the loss function to discourage the use of excessive parameter values.

Once the model has been trained, it can be used to make predictions on new, previously unseen data. Using the learned weights and bias, the input features are introduced into the logistic function, and the probability of the input belonging to the positive class is obtained. The input is classified as positive if the probability exceeds a predetermined threshold; otherwise, it is classified as negative.

That's a general overview of how logistic regression works. It is a basic yet effective algorithm for binary classification problems that is extensively employed in a variety of fields.

2.5 CNN Algorithm:

CNN, or Convolutional Neural Network, is a deep learning algorithm commonly employed for image recognition, object detection, and computer vision. It is particularly effective at visual data analysis because it can autonomously learn and derive image characteristics.

Here is a summary of how the CNN algorithm operates:

Convolutional Layer: The input image is first processed through one or more convolutional layers. Each layer consists of a set of teachable kernels or filters, which are small matrices. Convolution is a mathematical operation performed by these filters on the image input. Convolution involves the element-by-element multiplication of the filter with the corresponding region of the input image, followed by the addition of the resulting images. This operation generates a feature map that highlights particular patterns or features within the input image.

Non-linear Activation: After convolution, a non-linear activation function, such as ReLU (Rectified Linear Unit), is applied to the network to induce non-linearity. The activation function enables the network to acquire more intricate representations and facilitates the capture of complex

relationships.

Pooling Layer: After the activation function, down sampling pooling layers are applied to the feature maps. The operation of pooling reduces the spatial dimensions (width and height) of the feature maps while preserving vital data. The max pooling operation selects the greatest value within a specified region (such as a 2x2 window) and discards the rest. This down sampling decreases computational complexity and increases the network's robustness to input variations.

Fully Connected Layer: After multiple convolutional and pooling layers, the resulting feature maps are converted to a vector. The vector is then passed through dense layers or layers with complete connectivity. These layers are similar to those in conventional neural networks, in which each neuron is connected to each neuron in the layer beneath it. The layers with complete connectivity are taught to categorise the extracted features into particular categories.

Output Layer: After the final fully connected layer, there is an output layer with an appropriate activation function based on the current task. In classification assignments, for example, the softmax activation function is often used to derive probabilities for each class.

Training: CNN is trained using a large dataset containing examples with labels. During training, the network optimises its internal parameters (weights and biases) using backpropagation and gradient descent. The objective is to minimise a loss function that quantifies the difference between predicted outputs and actual labels. By iteratively modifying the parameters, the network learns to make accurate predictions.

Inference: Once the CNN has been trained, it can be used to make predictions on new, unobserved data. The input image is transmitted via the network, and depending on the specific assignment, the output layer provides the predicted class or other relevant information.

CNNs excel at automatically learning and extracting meaningful features from images, enabling them to achieve remarkable performance across a wide range of computer vision tasks.

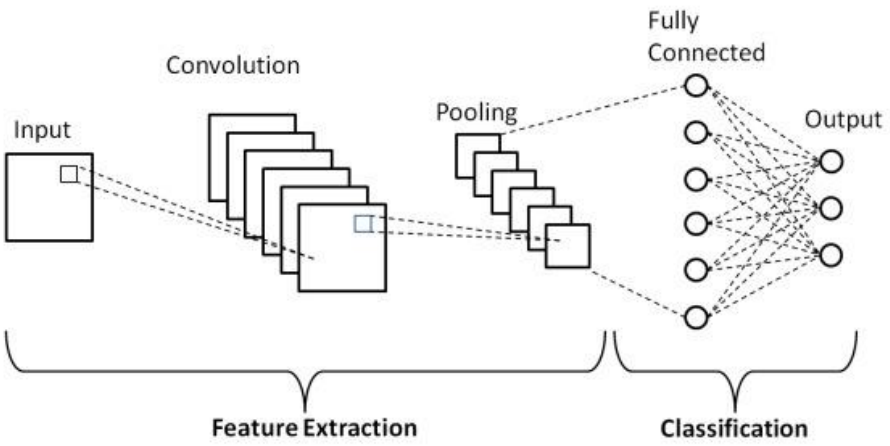


Fig 2.3: Schematic diagram of a basic convolutional neural network (CNN) architecture

2.6 DNN ALGORITHM:

Deep Neural Networks (DNNs) are a category of machine learning algorithms that are modelled after the structure and operation of the human brain. They consist of multiple layers of artificial neurons that are interconnected and organised hierarchically as nodes or units. Classification, regression, pattern recognition, and feature extraction from complex data are the primary applications of DNNs.

Here is a detailed explanation of how a typical DNN algorithm operates:

Data Preparation: The first stage in data preparation is to acquire and pre-process the data. This requires the collection of a labelled dataset consisting of input samples (features) and their corresponding outputs (labels). It may also be necessary to normalise or transform the data for optimal performance.

Network Architecture: The subsequent stage is to define the architecture of the DNN's network. This requires identifying the number of layers, the number of nodes in each layer, and their connections. The first layer, referred to as the input layer, receives the input data. The subsequent layers are referred to as hidden layers, and the output layer, which provides the predicted output, is the final layer.

Forward Propagation: In the forward propagation phase, input data is introduced into the DNN, and computations flow from the input layer through the hidden layers to the output layer. Each

node in a layer receives inputs from the nodes in the preceding layer, applies a transformation (activation function) to the weighted sum of the inputs, and then transmits the result to the nodes in the following layer.

Activation Function: Activation functions add non-linearity to the DNN, allowing it to learn complex patterns and relationships. Common activation functions include the sigmoid function, the hyperbolic tangent (tanh) function, and the rectified linear unit (ReLU) function.

Weight Adjustment: Initially, arbitrary values are assigned to the weights connecting the DNN's nodes. Throughout training, the DNN iteratively modifies these weights to minimise the gap between the predicted output and the actual objective output. This is accomplished through the backpropagation process.

Backpropagation: Backpropagation is a two-phase procedure that computes the gradients of the loss function with respect to the DNN's weights. Using a loss function (e.g., mean squared error or cross-entropy), the predicted output is compared to the objective output in the first phase. In the second phase, gradients are determined by propagating defects from the output layer to the input layer.

Weight Update: After calculating the gradients, the DNN's weights are modified using an optimisation algorithm such as stochastic gradient descent (SGD) or its derivatives. The weights are adjusted in the direction that minimises the loss function, with the learning rate determining the step size of the updates.

Training Iterations: Steps 3–7 are repeated for a number of iterations or epochs until the DNN converges to a state where the loss function is minimised and accurate predictions can be made. The number of iterations is dependent on the problem's complexity, the size of the dataset, and other variables.

Inference: Once the DNN has been trained, it can be used for inference and prediction on new, unobserved data. The input data is processed by the trained DNN using forward propagation, and the output layer provides the predicted output.

Evaluation: Depending on the nature of the problem, the efficacy of the DNN is evaluated using appropriate metrics such as accuracy, precision, recall, or mean squared error.

This is an overview of how a DNN algorithm operates at a high level. There are numerous techniques, architectures, and optimisations available for improving the efficacy and efficiency of DNNs in various applications.

2.7 LSTM Algorithm

Long-Short-Term Memory (LSTM) is a form of recurrent neural network (RNN) architecture that is designed to resolve the vanishing gradient problem in traditional RNNs. The vanishing gradient problem arises when training an RNN with lengthy sequences, making it challenging for the network to capture long-term dependencies.

LSTM networks incorporate a memory cell and several gating mechanisms, enabling the network to selectively remember or forget information over time. The fundamental concept underlying LSTM is the use of a memory cell, which is responsible for storing and retrieving information across lengthy sequences.

The following is a summary of how the LSTM algorithm operates:

Input and Output Gates: At each time step, the LSTM receives an input and determines which portions of the input to store and which portions to discard. This determination is made by the input gate, which accepts the input and the previous concealed state as inputs and generates a value between 0 and 1 for each constituent of the input.

Forget Gate: The LSTM must also determine which information in the memory cell to forget. This is done by the forget gate, which accepts the input and the previous hidden state as input and produces a value between 0 and 1 for each constituent of the memory cell. A value of 1 indicates that the information should be kept, whereas a value of 0 indicates that it should be discarded.

Memory Update: The memory cell alters its contents in response to the input gate and the neglect gate. The new information that must be stored in the memory cell is computed by multiplying the input gate value by a candidate value computed based on the input and the preceding concealed state. The neglect gate value is then multiplied by the current value of the memory cell to determine which information to retain.

Output Gate: The output gate receives the input and the previous concealed state as input and generates a value between 0 and 1 for each memory cell element. This value specifies which portions of the memory cell should be outputted to the next concealed state.

Hidden State: The hidden state represents the output of the LSTM at each time step. It is a composite of the present contents of memory cells that have been filtered by the output gate. In addition to the current input, the hidden state is supplied as an input to the next time phase.

By utilising these gating mechanisms and the memory cell, the LSTM is able to selectively store and retrieve information over long sequences, making it particularly effective for tasks requiring the capture of long-term dependencies, such as natural language processing, speech recognition, and time series prediction.

CHAPTER 3

3.1 MATERIALS AND METHODS

The general methodology and process by which the experimental study has been performed is shown in the Fig.

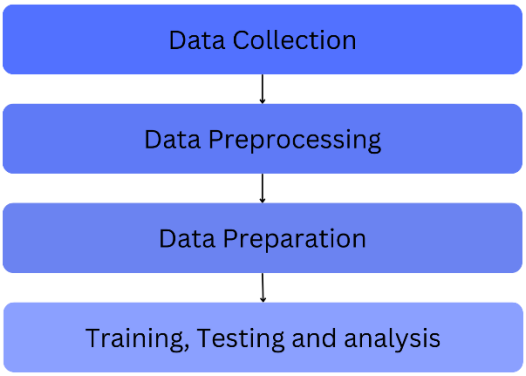


Fig 3.1: Major steps of the Proposed Method

3.2 COLLECTION OF DATA

We collected the metadata of ICLR submissions from an online peer-reviewing platform, OpenReview, from 2018 to 2021. The submissions were collected and filtered so that we could use the meta-reviews as a source of controllable text generation.

Table shows the statistics of data collected from each year. Initially, 7,894 submissions were collected. After filtering, 7,089 meta-reviews were retained with their corresponding 23,675 reviews.

In this study, we are using the dataset of ICLR 2020 and ICLR 2017 gathered from an individual which contains 1982 submission in the 2020 dataset and 367 submissions in the 2017 dataset with reviews, human generated paper reviews (3 for each paper) along with the meta review and the accept/reject status.

The 2020 dataset has 1473 Rejected papers and 508 accepted papers and 2017 dataset contains 229 rejected papers and 139 accepted papers.

3.3 DESCRIPTION OF DATA

The dataset contains human generated reviews along with the meta reviews and Accept/Reject status for the submitted papers.

Decision	meta-reviews	review 1	review 2	review 3
Reject	The authors propose a	This paper	This paper	This paper proposes an estimator
Accept	This paper investigates	This paper	Summary	This paper explores the effect of c
Reject	The authors propose a	Summary:	This paper	This paper claims that one only ne
Reject	The authors propose a	The paper	This paper	The authors study the problem of
Reject	This paper presents a c	This work	This paper	This paper presents a rotation-equ
Reject	The paper received scc	#ERROR!	A new onli	In this paper, the authors study the
Accept	This paper introduces a	[Summary	*Summary	This paper proposes a new magnit
Reject	The paper theoreticall	In this pap	This paper	The authors generalize Gaussian ra
Reject	The article studies ben	This work	This paper	The paper shows that under He ini
Accept	This paper studies the	This paper	This paper	The paper tackles the problem of r
Reject	The paper presents a s	This paper	- This pape	This submission introduces a semi-
Reject	This paper proposes a	*Summary	This paper	The authors present a framework
Reject	All reviewers agree tha	The autho	The paper	This paper conducts several experi
Reject	Thanks to the authors	The paper	This paper	Summary of the paper: The paper
Accept	The paper considers ar	In this wor	I haven't v	Authors propose a way to rectify t
Reject	This paper aims to esti	This paper	This paper	The problem addressed by this pap
Reject	We will certainly incor	The submi	Update af	This paper proposes a method for
Reject	The reviewers have rea	This work	In this wor	This paper provides convergence a
Reject	All three reviewers are	This work	This paper	This paper proposed a so-called d
Reject	This paper proposes ar	This work	In this pap	This paper presents several impro
Reject	The paper presents a g	Contributi	This paper	The motivation of this paper is to
Reject	Evolutionary strategies	Summary:	Summary:	Review of “Improving Evolutionary
Reject	This paper offers a pos	This paper	This paper	The paper introduces an off-policy
Reject	The reviewers found th	This paper	Summary:	#ERROR!
Reject	The authors propose a	The autho	This paper	The paper proposes a new metric
Accept	The paper proposes a	Summarize	I have rea	This paper proposes a new attenti
Accept	This paper presents a r	This work	This paper	This paper proposes a new metho
Reject	This paper proposes to	This paper	This paper	This paper proposed a topology-av

Fig 3.2: A small snippet of the dataset being used

3.4 METHODOLOGY

This research project's methodology consists of two main phases: the compilation of meta reviews and the prediction of academic paper decisions. First, text summarization algorithms are applied to each paper's review in the dataset in order to generate machine-generated meta-reviews. These summaries highlight the most significant evaluations and aspects of each paper and provide a concise summary

of the reviewers' opinions. To evaluate the veracity of the generated meta reviews, a comparison with the available human-generated meta reviews from the dataset is performed. Various metrics, such as Rouge scores and readability scores, are calculated to evaluate the similarity and coherence between machine-generated and human-generated meta reviews.In the second stage, the generated meta-reviews and acceptance/rejection status of the papers are used to train a set of deep learning models. These models include Convolutional Neural Networks (CNN), Deep Neural Networks (DNN), Long Short-Term Memory (LSTM), and Logistic Regression. Each model is trained on meta-analyses and their associated decisions to identify patterns and relationships that can be used to predict the acceptance or rejection of academic papers. Various metrics, such as accuracy, precision, recall, and F1-score, are employed to assess the efficacy of these models.We expect to identify the model with the maximum performance in terms of academic meta review generation and decision prediction by comparing the outcomes of multiple deep learning models. The chosen model should have a high degree of precision, effectively capture the essence of the meta-reviews, and provide precise predictions of paper acceptance or rejection. Through this detailed analysis, we expect to advance the generation and decision-making of automated meta-reviews in academic publishing.It is essential to note that throughout the methodology, rigorous data preprocessing and purification techniques are used to ensure the veracity and quality of the dataset. Moreover, hyperparameter optimisation and cross-validation techniques are implemented to optimise the effectiveness of deep learning models and prevent overfitting.

3.5 SOFTWARE AND HARDWARE REQUIREMENT

- **Command Prompt (CMD)** is a command line interpreter programme that may be found in the majority of Windows operating systems. It is used to implement orders inputted by the user. The majority of these commands perform complex administrative tasks, automate tasks with scripts and batch files, and troubleshoot or resolve Windows-specific issues. Command Prompt is also known as Command Shell, cmd Prompt, and by its filename, cmd.exe. Its official name is the Windows Command Processor.
- **Python**: Python is a widely applicable, versatile, and prominent programming language. It's great as a first language because it's concise and easy to read, and it's a good addition to any developer's toolbox because it can be used for everything from web development to programming development and logical applications. Python is a deciphered, high-level, widely

applicable programming language. Python's structure reasoning enhances code readability with its extensive use of significant whitespace. Its language constructs and item-centred methodology are designed to aid software developers in writing plain, intelligent code for projects of all sizes. Python is composed and collected in a progressive manner. Because of its extensive standard library, Python is frequently portrayed as a "batteries included" language. Python delimits squares with whitespace rather than undulating sections or watchwords. It possesses channel, map, and reduce capabilities, as well as list comprehensions, word references, sets, and generator articulations. Itertools and functools are two modules of the standard library that execute valuable Haskell and Standard ML apparatuses.

- **Pip Install** : Pip is an authentic standard framework for installing and managing Python programming modules. One can discover numerous combinations. The Python Package Index (PyPI) is the default repository for packages and associated conditions. Most Python distributions already have pip installed. Python 2.7.9 and later (on the python2 configuration) and Python 3.4 and later by default include Pip (pip3 for Python 3).
- **Google Collab**: It is a product of Google Research. Colab excels at machine learning, data analysis, and education. It permits anyone to construct and execute arbitrary Python code via a web browser.
- **Operating environment:** Operating system: Windows 7, Windows 8 and higher versions, Linux, MacOs Interpreter: Web application
- **Hardware Requirements:** Laptop / PC with 4 GB RAM Processor – i3 or higher version, AMD 3 or higher version Storage – 5 GB max Internet Connection 1 GHz speed

3.6 LIBRARIES USED

- **NLTK**: The Natural Language Toolkit (NLTK) is a Python library that facilitates the development of applications that perform natural language processing tasks such as tokenization, sentence segmentation, named entity recognition (NER), and semantic role labelling. Numerous researchers use it to execute these types of duties. NLTK is one of the most widely used open-source NLP libraries currently available. NLTK can be applied to any form of text data, such as text files (e.g., HTML files), email messages, Twitter posts, etc.
- **NUMPY**: NumPy is a Python library for array operations. It also has functions for linear algebra, the Fourier transform, and matrix operations.

NumPy was created by Travis Oliphant in 2005. It is an open-source project that you may freely utilise. NumPy, which stands for Numerical Python, concentrates on performing array operations in an efficient manner.

NumPy was created to leverage the capabilities of other open source libraries, such as SciPy, an open source library for scientific computation in Python. The objective is to develop a numerical system that can be utilised in production environments or by novice programmers with no prior experience with numerical computations.

With NumPy you can do things like:

-Work with arrays of numbers

-Perform matrix operations

-Convert between numeric input and textual output (e.g., ASCII)

- **Sumy** is a basic and user-friendly library that automatically summarises text. It has implemented a number of summarization algorithms, including TextRank, Luhn, Edmundson, LSA, LexRank, KL-Sum, SumBasic, and Reduction.

  It is used to extract knowledge from datasets by analysing the text to determine the most significant information in each sentence. This information can then be applied in a variety of ways, such as predicting future events based on past ones or generating recommendations based on user behaviour.

- **Textstat:** Textstat is a library that calculates statistics from text and is simple to use. It aids in determining the readability, difficulty, and grade level.

  Textstat employs words and sentences to determine the text's readability and grade level. The tool considers word length, sentence length, and word frequency to give you a clearer picture of the readability of your text.

- **spaCy:** SpaCy is a Python library for Natural Language Processing that offers an easy-to-use interface to the NLP algorithms it contains. It's built on top of Cython, meaning it's fast and efficient, and it's designed to be used in production environments.

- **BERT Extractive Summarizer:** Extractive Text Summarization with BERT Extractive Summarizer involves summarizing an article with the extracted key information via BERT natural language model. Extractive summarization is to provide decreased memory usage while protecting the content's value.

  The process of extractive summarization begins with the transformation of the input text into a feature vector, which is then applied to the machine learning model. The output of this procedure is a compact, machine- and human-readable representation of the original text. In

this process, both humans and machines are able to comprehend what the original text said, but only humans can access the information that was omitted from the output. Based on the context, the machine learning model identifies these missing terms.

- Scikit-learn: Scikit-learn is an open-source machine learning library for Python that provides a comprehensive collection of machine learning tools and algorithms. It offers a straightforward and effective method for implementing machine learning algorithms such as classification, regression, clustering, dimensionality reduction, and model evaluation. The modularity of the library makes it simple for users to traverse between the various modules for data pre-processing, feature extraction, model selection, training, and evaluation. With a consistent API and integration with popular Python libraries, Scikit-Learn enables users to seamlessly work with numerical data representations and extend its functionalities. It is a valuable resource for both novice and experienced practitioners in the field of machine learning due to its active community and extensive documentation.

3.7 SAMPLE CODES

3.7.1 LexRank

- Library Installation

```
pip install nltk
pip install sumy
```

- Importing packages from the library

```
from sumy.parsers.plaintext import PlaintextParser
from sumy.nlp.tokenizers import Tokenizer
import nltk
```

- Input file containing the reviews

```
file = "iclr41.txt"
```

- Pre-processing the text

```
nltk.download('punkt')
parsers = PlaintextParser.from_file(file, Tokenizer("english"))
from sumy.summarizers.lex_rank import LexRankSummarizer
summarizer = LexRankSummarizer()
```

- Printing the output

```
#Summarize the document with 20 sentences
summary = summarizer(parsers.document, 20)
for sentence in summary:
 print(sentence)
```

3.7.2 TextRank

- Library Installations

```
pip install nltk
pip install sumy
```

- Importing packages from the library

```
from sumy.parsers.plaintext import PlaintextParser
from sumy.nlp.tokenizers import Tokenizer
import nltk
```

- Input file containing the reviews

```
file = "iclr41.txt"
```

- Pre-processing the text

```
nltk.download('punkt')
parsers = PlaintextParser.from_file(file, Tokenizer("english"))
from sumy.summarizers.text_rank import TextRankSummarizer
summarizer = TextRankSummarizer()
```

- Printing the output

```
#Summarize the document with 20 sentences
summary = summarizer(parsers.document, 20)
for sentence in summary:
 print(sentence)
```

3.7.3 BERT

- Installing libraries

```
pip install bert-extractive-summarizer
pip install neuralcoref
pip install transformers
pip install spacy
```

- Importing Libraries

```
from summarizer import Summarizer
from pprint import pprint
```

- Input text file and pre-processing

```
with open("iclr21.txt", 'r', encoding="UTF-8") as file:
    data = file.read().replace('\n', '')
data = data.replace("\ufeff", "")
data[1000:5800]
model = Summarizer()
result = model(data[1000:25800], num_sentences=20,
min_length=60)
full = ''.join(result)
```

- Print summary

```
pprint(full)
```

3.7.4 ROUGE Score

- Installing Library

```
pip install rouge
```

- Importing Library

```
from rouge import Rouge
```

- Providing Input for generated review and Meta-review

```
model_out = ['''''']

reference = ['''''']
```

- Calculating scores

```
rouge = Rouge()

rouge.get_scores(model_out, reference)
```

3.7.5 READABILITY Score

- Installing Libraries

```
pip install textstat
```

- Importing Library

```
import textstat
```

- Providing input

```
text = ''''''
```

- Calculating scores

```
textstat.flesch_reading_ease(text)
score = textstat.automated_readability_index(text)
print(score)
textstat.flesch_kincaid_grade(text)
```

3.7.6 Automated Lexrank Text Summerizer:

- The code starts by importing necessary modules and functions:

```
import openpyxl
from openpyxl.utils import get_column_letter
from sumy.summarizers.lex_rank import LexRankSummarizer
from sumy.nlp.tokenizers import Tokenizer
from sumy.parsers.plaintext import PlaintextParser
from sumy.utils import get_stop_words
```

- The main function summarize_excel_file is defined with several parameters:

```
def summarize_excel_file(file_path, cell_range,
sentences_count=5, threshold=0.1, output_dir="."):
```

- The function starts by loading the Excel file using openpyxl:

```
wb = openpyxl.load_workbook(file_path)

ws = wb.active
```

- The cell values within the specified range are extracted from the worksheet and stored in the cell_values list:

```
cell_values = [cell.value for row in ws[cell_range] for cell
in row]
```

- The cell values are concatenated into a single string called text:

```
text = "\n".join(str(cell) for cell in cell_values)
```

- Next, the code sets up the summarizer using LexRank algorithm:

```
LANGUAGE = "english"
tokenizer = Tokenizer(LANGUAGE)
parser = PlaintextParser.from_string(text, tokenizer)
summarizer = LexRankSummarizer()
```

- The summarizer parameters are set:

```
summarizer.stop_words = get_stop_words(LANGUAGE)
summarizer.threshold = threshold
```

- The code generates the summary by applying the summarizer to the parsed document:

```
summary = ""
for sentence in summarizer(parser.document, sentences_count):
    summary += str(sentence) + " "
```

- The summary is saved to an Excel file:

```
output_file = f"{output_dir}/metareview.xlsx"
try:
    wb_output = openpyxl.load_workbook(output_file)
except FileNotFoundError:
```

```
    wb_output = openpyxl.Workbook()
ws_output = wb_output.active
column_letter = 'A'
row_number = ws_output.max_row + 1
cell = ws_output[f"{column_letter}{row_number}"]
cell.value = summary.strip()
wb_output.save(output_file)
```

- **The summary is also saved to a separate text file:**

```
output_path = f"{output_dir}/{file_path.split('/')[-
1].replace('.xlsx', '')}_lexrank_summary
```

3.7.7 Logistic Regression

```
import pandas as pd
import nltk
from sklearn.feature_extraction.text import CountVectorizer,
TfidfTransformer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

# load the meta review data into a Pandas dataframe
meta_review_df = pd.read_csv('data2020.csv')

# preprocess the text data using NLTK
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')

from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize

lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

def preprocess_text(text):
    tokens = word_tokenize(text.lower())
    tokens = [token for token in tokens if token.isalpha() and
token not in stop_words]
    tokens = [lemmatizer.lemmatize(token) for token in tokens]
    return ' '.join(tokens)
```

```
meta_review_df['processed_text'] =
meta_review_df['review_text'].apply(preprocess_text)

# build a bag-of-words model and a TF-IDF transformer
vectorizer = CountVectorizer()
transformer = TfidfTransformer()

X = vectorizer.fit_transform(meta_review_df['processed_text'])
X = transformer.fit_transform(X)

y = meta_review_df['acceptance'].astype(int)

# split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# train a logistic regression model on the training set
clf = LogisticRegression(random_state=42)
clf.fit(X_train, y_train)

# evaluate the model on the testing set
accuracy = clf.score(X_test, y_test)
print('Accuracy:', accuracy)
```

3.7.8 DNN Model

```
import pandas as pd

import nltk

from sklearn.feature_extraction.text import CountVectorizer,
TfidfTransformer

from sklearn.model_selection import train_test_split

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras.callbacks import EarlyStopping


# Load the meta review data into a Pandas dataframe

meta_review_df = pd.read_csv('data2020.csv')
```

```
# Preprocess the text data using NLTK
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')

from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize

lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

def preprocess_text(text):
    if isinstance(text, str):
        tokens = word_tokenize(text.lower())
        tokens = [token for token in tokens if token.isalpha()
and token not in stop_words]
        tokens = [lemmatizer.lemmatize(token) for token in
tokens]
        return ' '.join(tokens)
    else:
        return ''


meta_review_df['processed_text'] =
meta_review_df['review_text'].apply(preprocess_text)

# Split the data into training and test sets
X_train, X_test, y_train, y_test =
train_test_split(meta_review_df['processed_text'],
meta_review_df['acceptance'], test_size=0.2, random_state=42)

# Vectorize the text data using CountVectorizer and
TfidfTransformer
```

```
vectorizer = CountVectorizer()
transformer = TfidfTransformer()

X_train = vectorizer.fit_transform(X_train)
X_train = transformer.fit_transform(X_train)

X_test = vectorizer.transform(X_test)
X_test = transformer.transform(X_test)

# Convert to dense arrays
X_train = X_train.toarray()
X_test = X_test.toarray()

# Build the DNN model
input_dim = X_train.shape[1]

model = Sequential()
model.add(Dense(128, input_dim=input_dim, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

# Train the model
early_stopping = EarlyStopping(patience=3,
restore_best_weights=True)
model.fit(X_train, y_train, validation_data=(X_test, y_test),
epochs=10, callbacks=[early_stopping])

# Evaluate the model
_, accuracy = model.evaluate(X_test, y_test)
print('Accuracy:', accuracy)
```

```
# Save the trained model
model.save('acceptance_model.h5')
```

3.7.9 CNN Model

```
import pandas as pd
import numpy as np
import nltk
from sklearn.feature_extraction.text import CountVectorizer,
TfidfTransformer
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Conv1D,
GlobalMaxPooling1D, Dense
from tensorflow.keras.callbacks import EarlyStopping

# Load the meta review data into a Pandas dataframe
meta_review_df = pd.read_csv('data2020.csv')

# Drop rows with missing or NaN values in the 'review_text' column
meta_review_df = meta_review_df.dropna(subset=['review_text'])

# Preprocess the text data using NLTK
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')

from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
```

```
lemmatizer = WordNetLemmatizer()

stop_words = set(stopwords.words('english'))


def preprocess_text(text):

    if isinstance(text, str):  # Check if the text is of type
string

        tokens = word_tokenize(text.lower())

        tokens = [token for token in tokens if token.isalpha() and
token not in stop_words]

        tokens = [lemmatizer.lemmatize(token) for token in tokens]

        return ' '.join(tokens)

    else:

        return ''


meta_review_df['processed_text'] =
meta_review_df['review_text'].apply(preprocess_text)


# Split the data into training and test sets

X_train, X_test, y_train, y_test =
train_test_split(meta_review_df['processed_text'],
meta_review_df['acceptance'], test_size=0.2, random_state=42)


# Vectorize the text data using CountVectorizer and
TfidfTransformer

vectorizer = CountVectorizer()

transformer = TfidfTransformer()


X_train = vectorizer.fit_transform(X_train)

X_train = transformer.fit_transform(X_train)


X_test = vectorizer.transform(X_test)

X_test = transformer.transform(X_test)
```

```
# Pad sequences to a fixed length

max_sequence_length = max([len(text.split()) for text in
meta_review_df['processed_text']])

X_train = pad_sequences(X_train.toarray(),
maxlen=max_sequence_length)

X_test = pad_sequences(X_test.toarray(),
maxlen=max_sequence_length)


# Build the CNN model

vocab_size = len(vectorizer.vocabulary_)

embedding_dim = 100

filters = 64

kernel_size = 3


model = Sequential()

model.add(Embedding(vocab_size, embedding_dim,
input_length=max_sequence_length))

model.add(Conv1D(filters, kernel_size, activation='relu'))

model.add(GlobalMaxPooling1D())

model.add(Dense(1, activation='sigmoid'))


model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])


# Train the model

early_stopping = EarlyStopping(patience=3,
restore_best_weights=True)

model.fit(X_train, y_train, validation_data=(X_test, y_test),
epochs=10, callbacks=[early_stopping])


# Evaluate the model

_, accuracy = model.evaluate(X_test, y_test)

print('Accuracy:', accuracy)
```

```
# Save the trained model
model.save('acceptance_model.h5')
```

3.7.10 LSTM Model

```
import pandas as pd
import numpy as np
import nltk
from sklearn.feature_extraction.text import CountVectorizer,
TfidfTransformer
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense
from tensorflow.keras.callbacks import EarlyStopping

# Load the meta review data into a Pandas dataframe
meta_review_df = pd.read_csv('data2020.csv')

# Drop rows with missing or NaN values in the 'review_text' column
meta_review_df = meta_review_df.dropna(subset=['review_text'])

# Preprocess the text data using NLTK
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')

from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
```

```
lemmatizer = WordNetLemmatizer()

stop_words = set(stopwords.words('english'))


def preprocess_text(text):

    if isinstance(text, str):  # Check if the text is of type
string

        tokens = word_tokenize(text.lower())

        tokens = [token for token in tokens if token.isalpha() and
token not in stop_words]

        tokens = [lemmatizer.lemmatize(token) for token in tokens]

        return ' '.join(tokens)

    else:

        return ''


meta_review_df['processed_text'] =
meta_review_df['review_text'].apply(preprocess_text)


# Split the data into training and test sets

X_train, X_test, y_train, y_test =
train_test_split(meta_review_df['processed_text'],
meta_review_df['acceptance'], test_size=0.2, random_state=42)


# Vectorize the text data using CountVectorizer and
TfidfTransformer

vectorizer = CountVectorizer()

transformer = TfidfTransformer()


X_train = vectorizer.fit_transform(X_train)

X_train = transformer.fit_transform(X_train)

X_test = vectorizer.transform(X_test)

X_test = transformer.transform(X_test)

# Pad sequences to a fixed length
```

```
max_sequence_length = max([len(text.split()) for text in
meta_review_df['processed_text']])

X_train = pad_sequences(X_train.toarray(),
maxlen=max_sequence_length)

X_test = pad_sequences(X_test.toarray(),
maxlen=max_sequence_length)


# Build the RNN-LSTM model

vocab_size = len(vectorizer.vocabulary_)

embedding_dim = 100

hidden_units = 64


model = Sequential()

model.add(Embedding(vocab_size, embedding_dim,
input_length=max_sequence_length))

model.add(LSTM(hidden_units))

model.add(Dense(1, activation='sigmoid'))


model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])


# Train the model

early_stopping = EarlyStopping(patience=3,
restore_best_weights=True)

model.fit(X_train, y_train, validation_data=(X_test, y_test),
epochs=10, callbacks=[early_stopping])


# Evaluate the model

_, accuracy = model.evaluate(X_test, y_test)

print('Accuracy:', accuracy)


# Save the trained model

model.save('acceptance_model.h5')
```

CHAPTER 4

4.1 RESULTS

From the ICLR 2020 dataset, we chose 40 papers for the study. Then we created a text file with the three reviews for each paper and fed the three models to it (Lexrank, TextRank and BERT). We calculated the ROUGE score for each summary after receiving the summaries from each model, taking into account the provided meta-reviews from the dataset. Additionally, using the generated summaries, we computed the readability scores (Flesch reading ease, Readability index, and Flesch Kincaid grade).

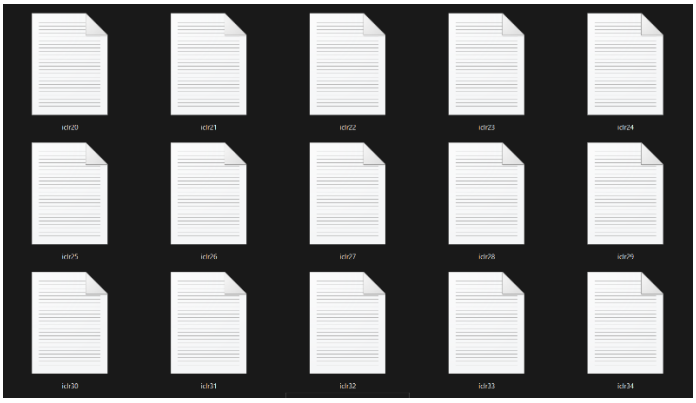


Fig 4.1: A small snippet of the review files

4.1.1 Comparison Between Human-Generated and Machine Generated Meta Review

Human-Generated Meta Review:

The authors propose a novel metric to detect distributional discrepancy for text generation models and argue that these can be used to explain the failure of GANs for language generation tasks. The reviewers found significant deficiencies with the paper, including: 1) Numerous grammatical errors and typos, that make it difficult to read the paper. 2) Mischaracterization of prior work on neural language models, and failure to compare with standard distributional discrepancy measures studied in prior work (KL, total variation, Wasserstein etc.). Further, the necessity of the complicated procedure derived by the authors is not well-justified. 3) Failure to run experiments on standard benchmarks for image generation (which are much better studied applications of GANs) and confirm the superiority of the proposed metrics relative to standard baselines. The reviewers were agreed on the rejection decision and the authors did not participate in the rebuttal phase. I therefore recommend rejection.

Machine Generated Meta-Review:

This paper argues that text generated by existing neural language models are not as good as real text and proposes two metric functions to measure the distributional difference between real text and generated text. The proposed metrics are tried on language GANs but fail to produce any improvement. Major issues: This manuscript is poorly organized, and the introduction is not well-written. It's true that generating text from random noise vector remains a challenging problem, but sequence-to-sequence models for machine translation and question answering have achieved tremendous successes. 2: "We analysis…", "could be find…", pp 3: "equation 8" should be "equation 9"...The proposed

metrics are also questionable. Eq. 3 on page 2 holds for any x sampled from the distribution, not just for a single data point. This paper mixes text generation and proposed metrics together. The claimed failure experiments make the proposed metrics even more questionable. This paper proposes two metrics to measure the discrepancy between generated text and real text, based on the discriminator score in GANs. Empirically, it shows that text generated by current text generation methods is still far from human-generated text, as measured by the proposed metric. The writing is a bit rough so sometimes it's hard to figure out what has been done. It's also unclear how the proposed metrics compare to simply using the discriminator for evaluation. Approach: - The proposed metric essentially relies on the learned discriminator to measure the closeness of generated text vs real text, based on the strong assumption that the learned discriminator is near-optimal. It has been previously shown that learning a classifier from generated and real text does not generalize well (Lowe et al, 2017, Chaganty et al, 2018). This paper proposes an estimator to quantify the difference in distributions between real and generated text based on a classifier that discriminates between real vs generated text. As such, the authors should demonstrate a power analysis of their test to detect differences between real vs generated text and show this new test is better than tests based on existing discrepancy measures. - The authors claim training a generator to minimize their proposed divergence is superior to a standard language GAN. Minor Comments: - x needs to be defined before equation (1).

4.1.2 LexRank

After providing the review files into the LexRank model and generating the reviews we calculated the ROUGE scores and the READABILITY scores and got the following results:

Paper no.	Rouge-1			Rouge-2			Rouge-L		
	r	p	f	r	p	f	r	p	f
1	0.37623	0.1719	0.2360	0.1118	0.04481	0.06399	0.3564	0.1628	0.2236
2	0.5797	0.1532	0.2424	0.0909	0.01834	0.0305	0.5072	0.1340	0.2121
3	0.2857	0.0984	0.1464	0.04273	0.01070	0.01712	0.26373	0.09090	0.1352
4	0.6470	0.0718	0.1294	0.175	0.0123	0.02306	0.6176	0.06862	0.1235
5	0.4234	0.1747	0.2473	0.1125	0.03930	0.05825	0.3603	0.14869	0.2105
6	0.4107	0.09745	0.15753	0.05633	0.01020	0.01727	0.32142	0.07627	0.12328
7	0.5	0.06896	0.12121	0.12195	0.01240	0.0225	0.5	0.0689	0.12121
8	0.4313	0.18965	0.26347	0.15972	0.06149	0.08880	0.4215	0.18534	0.25748
9	0.5660	0.1287	0.20979	0.19696	0.032418	0.05567	0.56603	0.12875	0.20979
10	0.45205	0.10610	0.17187	0.08411	0.01768	0.029220	0.36986	0.08681	0.14062
11	0.37681	0.1150	0.17627	0.09574	0.02295	0.03703	0.33333	0.10176	0.15593
12	0.33333	0.17142	0.22641	0.075	0.031578	0.044444	0.31111	0.16	0.2113
13	0.5	0.08121	0.13973	0.11111	0.01212	0.02185	0.5	0.08121	0.13973
14	0.43478	0.09174	0.151515	0.08620	0.014326	0.02457	0.43478	0.091743	0.15151
15	0.45454	0.07317	0.12605	0.15	0.017543	0.03141	0.39393	0.06341	0.10924
16	0.44705	0.1484	0.22287	0.15966	0.04042	0.064516	0.4	0.13281	0.19941
17	0.2888	0.2888	0.20259	0.03157	0.01435	0.01973	0.25185	0.136	0.17662
18	0.43478	0.04878	0.08771	0.13043	0.00872	0.01634	0.43478	0.0487	0.0877
19	0.42857	0.02912	0.05454	0	0	0	0.42857	0.02912	0.05454
20	0.464788	0.12547	0.19760	0.05681	0.01131	0.01886	0.42253	0.11406	0.17964
21	0.3720	0.1264	0.1887	0.11016	0.03201	0.0496	0.3372	0.11462	0.17109

22	0.3356	0.1731	0.2284	0.07373	0.03524	0.0476	0.2945	0.1519	0.2004
23	0.4528	0.08450	0.14243	0.13846	0.01898	0.03339	0.4339	0.0809	0.13649
24	0.4634	0.07630	0.1310	0.0652	0.0071	0.01287	0.4634	0.07630	0.13103
25	0.55555	0.16949	0.2597	0.16666	0.03740	0.06109	0.48611	0.14830	0.22727
26	0.53125	0.08673	0.1491	0.03125	0.003508	0.006309	0.5	0.0816	0.1403
27	0.36363	0.07881	0.12955	0.03703	0.0065	0.01108	0.31818	0.0689	0.1133
28	0.41666	0.0684	0.11764	0.01694	0.00202	0.003	0.375	0.06164	0.1058
29	0.4382	0.13636	0.20799	0.1681	0.04112	0.06608	0.4044	0.12587	0.19199
30	0.5820	0.2335	0.3333	0.23913	0.08906	0.12979	0.5522	0.2215	0.31623
31	0.3571	0.1345	0.1954	0.0614	0.0193	0.02941	0.3095	0.1165	0.1693
32	0.31818	0.1359	0.1904	0.025	0.0095	0.01379	0.2613	0.1116	0.1564
33	0.2962	0.0717	0.1155	0.0845	0.0162	0.0272	0.2777	0.0672	0.1083
34	0.34666	0.1097	0.1666	0.07777	0.0179	0.0292	0.32	0.1012	0.1538
35	0.3787	0.08771	0.1424	0.0864	0.0145	0.0248	0.3181	0.0736	0.1196
36	0.317	0.0575	0.0973	0.0697	0.0084	0.0149	0.2926	0.053	0.0898
37	0.2903	0.1075	0.1569	0.0422	0.01503	0.0221	0.258	0.0956	0.1395
38	0.32	0.1016	0.1543	0.07	0.0186	0.02947	0.2666	0.0847	0.1286
39	0.2941	0.0867	0.1339	0.0327	0.0079	0.0127	0.2941	0.0867	0.1339
40	0.4153	0.095	0.1547	0.0681	0.0119	0.0203	0.3692	0.0845	0.1375

Table 4.1: ROUGE score table of LexRank

Paper no.	Readability score		
	Flesch reading ease	Index	Flesch grade level
1	43.83	13.3	11.8
2	49.25	14.4	11.8
3	44.78	16.5	13.5
4	41.53	17.2	14.8
5	53.95	14.2	12.1
6	48.54	13.5	12.1
7	41.19	15.4	12.9
8	42.0	14.4	12.5
9	44.37	17.1	13.7
10	36.63	16.7	14.6
11	41.5	15.1	12.7
12	59.03	12.8	10.1
13	54.32	10.9	9.9
14	52.9	12.4	10.4
15	43.97	16.5	13.9
16	53.75	13.5	12.2
17	41.7	13.7	12.7
18	54.02	12.0	10.0
19	50.36	13.4	11.4
20	55.17	15.1	11.6
21	44.17	17.1	13.8
22	46.61	15.1	12.8
23	56.18	13.0	11.2
24	39.87	14.2	13.4
25	52.19	11.9	10.7
26	56.25	10.1	9.1

27	52.6	13.2	10.5
28	46.1	16.3	13.0
29	40.58	14.5	13.1
30	46.47	12.3	10.8
31	51.28	13.2	11.1
32	52.9	12.6	10.4
33	40.08	14.4	13.3
34	41.6	14.2	12.7
35	35.1	16.8	15.2
36	40.79	15	13
37	51.18	13	11.1
38	51.99	12	10.8
39	41.29	14.9	12.8
40	53.65	15	12.2

Table 4.2: Readability Score table LexRank

4.1.3 TextRank

After providing the review files into the TextRank model and generating the reviews we calculated the ROUGE scores and the READABILITY scores and got the following results:

Paper no.	Rouge-1			Rouge-2			Rouge-L		
	r	p	f	r	p	f	r	p	f
1	0.3762	0.1645	0.2289	0.0979	0.0373	0.0540	0.3465	0.1515	0.2108
2	0.5362	0.1316	0.2114	0.0681	0.0130	0.0218	0.4927	0.1209	0.1942
3	0.3186	0.1050	0.1580	0.0769	0.0183	0.0296	0.3076	0.1014	0.1525
4	0.67647	0.072327	0.13068	0.175	0.0117	0.02194	0.67647	0.072327	0.13068
5	0.42342	0.170289	0.24289	0.1125	0.038216	0.05705	0.36036	0.14492	0.20671
6	0.42105	0.1	0.1616	0.05555	0.01010	0.01709	0.33333	0.07916	0.12794
7	0.56756	0.075	0.13249	0.09523	0.00917	0.01673	0.5135	0.06785	0.11987
8	0.42718	0.1880	0.26112	0.15862	0.06117	0.08829	0.41747	0.18376	0.25519
9	0.5555	0.1287	0.20905	0.19402	0.032418	0.05555	0.55555	0.12875	0.20905
10	0.48648	0.1132	0.1836	0.11111	0.02290	0.03797	0.43243	0.10062	0.16326
11	0.3857	0.11297	0.17475	0.10526	0.02403	0.03913	0.34285	0.10041	0.1553
12	0.34065	0.16756	0.22463	0.07438	0.02941	0.04215	0.31868	0.15675	0.21014
13	0.48484	0.07511	0.13008	0.16216	0.01657	0.03007	0.45454	0.07042	0.12195
14	0.4255	0.08888	0.14705	0.08474	0.01373	0.02364	0.42553	0.08888	0.14705
15	0.44117	0.06944	0.11999	0.14634	0.01652	0.02970	0.38235	0.06018	0.10399
16	0.45348	0.14716	0.22222	0.175	0.04375	0.06999	0.38372	0.12452	0.18803
17	0.27205	0.14285	0.18734	0.03141	0.01405	0.01941	0.23529	0.12355	0.16202
18	0.43478	0.04716	0.08510	0.13043	0.00824	0.01550	0.43478	0.04716	0.085106

19	0.42857	0.02912	0.05454	0	0	0	0.42857	0.02912	0.05454
20	0.47887	0.12592	0.19941	0.07954	0.01580	0.02636	0.39436	0.10370	0.16422
21	0.37209	0.12648	0.18879	0.11016	0.03201	0.04961	0.33720	0.11462	0.17109
22	0.32653	0.14906	0.20469	0.06422	0.02692	0.03794	0.28571	0.13043	0.17910
23	0.42592	0.08273	0.13855	0.13636	0.01931	0.03383	0.38888	0.07553	0.12650
24	0.45238	0.07450	0.12794	0.06382	0.00691	0.01247	0.45238	0.07450	0.12794
25	0.51388	0.14285	0.22356	0.21111	0.04185	0.06985	0.45833	0.12741	0.19939
26	0.53125	0.08415	0.14529	0.03125	0.00335	0.00606	0.46875	0.07425	0.12820
27	0.36363	0.07881	0.12955	0.03703	0.00651	0.01108	0.31818	0.06896	0.11336
28	0.47916	0.07443	0.12885	0.05084	0.00548	0.00990	0.41666	0.06472	0.11204
29	0.46067	0.12424	0.19570	0.15929	0.03243	0.05389	0.43820	0.11818	0.18615
30	0.58208	0.23353	0.33333	0.23913	0.08906	0.12979	0.55223	0.22155	0.31623
31	0.30722	0.21428	0.25247	0.06637	0.03846	0.04870	0.28915	0.20168	0.23762
32	0.38095	0.14285	0.20779	0.11403	0.03768	0.05664	0.35714	0.13392	0.19480
33	0.39772	0.15695	0.22508	0.05833	0.01897	0.02862	0.36363	0.14349	0.20578
34	0.42592	0.09704	0.15807	0.21126	0.03856	0.06521	0.35185	0.08016	0.13058
35	0.53333	0.14035	0.22222	0.17777	0.03319	0.05594	0.53333	0.14035	0.22222
36	0.46268	0.13716	0.21160	0.10975	0.02521	0.04100	0.43283	0.12831	0.19795
37	0.36585	0.05882	0.10135	0.09302	0.01002	0.01809	0.34146	0.05490	0.09459
38	0.59139	0.23504	0.33639	0.31690	0.11688	0.17077	0.55913	0.22222	0.31804
39	0.4	0.17341	0.24193	0.13	0.05138	0.07365	0.38666	0.16763	0.23387
40	0.52941	0.07541	0.13202	0.13114	0.01335	0.02424	0.49019	0.06983	0.12224

Table 4.3: ROUGE score table of TextRank

Paper no.	Readability score		
	Flesch reading ease	Index	Flesch grade level
1	42.41	14.0	12.4
2	48.23	14.8	12.2
3	43.36	17.1	14.1
4	39.71	18.0	15.5
5	53.55	14.2	12.3
6	48.54	13.5	12.1
7	40.48	15.4	13.1
8	42.0	14.4	12.5
9	44.37	17.1	13.7
10	35.91	17.0	14.9
11	39.97	15.9	13.3
12	59.03	12.8	10.1

13	54.32	10.2	9.9
14	52.09	13.3	10.7
15	43.97	16.5	13.9
16	53.04	13.9	12.4
17	41.09	14.1	12.9
18	53.71	12.0	10.1
19	50.36	13.4	11.4
20	56.89	14.1	11.0
21	44.17	17.1	13.8
22	43.16	17.5	14.2
23	56.39	12.9	11.2
24	47.42	14.5	12.5
25	49.25	12.9	11.8
26	55.64	10.9	9.4
27	52.6	13.2	10.5
28	39.5	19.5	15.6
29	36.63	16.6	14.6
30	46.47	12.3	10.8
31	42.0	14.1	12.5
32	51.07	13.7	11.1
33	40.08	14.4	13.3
34	41.6	14.2	12.7
35	35.1	16.8	15.2
36	40.79	15.0	13.0
37	51.28	12.8	11.1
38	42.92	12.5	12.2
39	41.29	14.9	12.8
40	38.49	19.1	16.0

Table 4.4: Readability Score table TextRank

4.1.4 BERT

After providing the review files into the BERT model and generating the reviews we calculated the ROUGE scores and the READABILITY scores and got the following results:

Paper no.	Rouge-1			Rouge-2			Rouge-L		
	r	p	f	r	p	f	r	p	f
1	0.3168	0.1403	0.1945	0.0699	0.0294	0.04140	0.2970	0.1315	0.1823
2	0.4927	0.1387	0.2165	0.0568	0.0128	0.0209	0.4492	0.1265	0.1974
3	0.2527	0.0916	0.1345	0.0427	0.0115	0.0181	0.21978	0.0796	0.1169
4	0.7058	0.0662	0.1212	0.2	0.0118	0.0224	0.6764	0.0635	0.1161

5	0.2882	0.1481	0.1957	0.0625	0.0282	0.0389	0.2252	0.1157	0.1529
6	0.2631	0.08333	0.1265	0.0138	0.0037	0.0058	0.1578	0.05	0.0759
7	0.5555	0.07692	0.1351	0.0975	0.0099	0.01797	0.5277	0.0730	0.1283
8	0.3921	0.2409	0.2985	0.1180	0.07112	0.0887	0.3431	0.2108	0.2611
9	0.4629	0.1373	0.2118	0.1492	0.03558	0.0574	0.4629	0.1373	0.2118
10	0.2328	0.0934	0.1333	0.0560	0.0213	0.03092	0.2191	0.0879	0.1254
11	0.3285	0.1095	0.1642	0.0947	0.02631	0.0411	0.3	0.1	0.1499
12	0.3	0.2389	0.2660	0.0416	0.03125	0.03571	0.2555	0.2035	0.2266
13	0.5	0.08791	0.1495	0.1666	0.0212	0.0376	0.5	0.0879	0.1495
14	0.3260	0.0931	0.1449	0.0689	0.0173	0.0277	0.3260	0.0931	0.1449
15	0.4545	0.0847	0.1428	0.125	0.0186	0.0324	0.3939	0.07344	0.1238
16	0.3411	0.1266	0.1847	0.1008	0.0298	0.0460	0.2941	0.1091	0.1592
17	0.2296	0.1435	0.1766	0.0210	0.0119	0.0152	0.2148	0.1342	0.1652
18	0.3913	0.0511	0.0904	0.1304	0.0110	0.02033	0.3913	0.0511	0.0904
19	0.3571	0.03703	0.06711	0.0	0.0	0.0	0.3571	0.03703	0.0671
20	0.4084	0.0963	0.1559	0.0681	0.01176	0.0200	0.3802	0.0897	0.1451
21	0.2790	0.13259	0.17977	0.0593	0.02611	0.0362	0.2325	0.11049	0.1498
22	0.30821	0.15570	0.20689	0.0414	0.01973	0.02674	0.27397	0.138408	0.18390
23	0.358490	0.10982	0.168141	0.076923	0.01838	0.02967	0.30188	0.09248	0.14159
24	0.28301	0.06944	0.1115	0.0615	0.01179	0.01980	0.24528	0.06018	0.09665
25	0.34146	0.05490	0.09459	0.04347	0.00467	0.00843	0.31707	0.05098	0.08783
26	0.23611	0.1164	0.1559	0.04444	0.01877	0.02640	0.23611	0.11643	0.15596
27	0.15625	0.03937	0.06289	0	0	0	0.15625	0.039370	0.06289
28	0.36363	0.04863	0.0857	0.0370	0.0035	0.0064	0.25	0.0334	0.0589
29	0.3125	0.05639	0.0955	0.0508	0.0069	0.0122	0.2916	0.0526	0.0891
30	0.2247	0.2409	0.2325	0.0265	0.0280	0.02727	0.2022	0.2168	0.2093
31	0.4477	0.1115	0.1785	0.07608	0.01631	0.0268	0.4179	0.1040	0.1666
32	0.1867	0.1448	0.1631	0.0265	0.01807	0.0215	0.1686	0.1308	0.1473
33	0.25	0.1304	0.1714	0.04385	0.0209	0.0283	0.2142	0.1118	0.1469
34	0.29545	0.1585	0.2063	0.01666	0.0085	0.01129	0.2613	0.1402	0.1825
35	0.2777	0.0678	0.10909	0.0845	0.01648	0.02758	0.2592	0.0633	0.10181
36	0.2933	0.1419	0.19130	0.0444	0.0176	0.02531	0.28	0.13548	0.18260
37	0.3066	0.1173	0.1697	0.0333	0.0105	0.01599	0.28	0.1071	0.1549
38	0.28787	0.1043	0.1532	0.0246	0.0072	0.01126	0.2575	0.0934	0.1370
39	0.2682	0.12087	0.16666	0	0	0	0.2439	0.10989	0.1515
40	0.4680	0.24175	0.3188	0.2167	0.1131	0.14868	0.4255	0.21978	0.28985

Table 4.5: ROUGE score table of BERT

Paper no.	Readability		
	Flesch reading ease	index	Flesch Kincaid grade
1	45.76	14.5	11.1
2	42.0	15.1	12.5
3	44.07	17.3	13.8
4	46.71	16.0	12.8
5	56.79	13.9	11.0
6	44.44	12.8	11.6
7	41.4	16.2	12.8
8	40.28	17.6	13.2
9	36.02	19.3	14.8
10	45.69	16.8	13.2
11	39.26	17.0	13.6
12	60.55	13.1	9.6

13	55.13	11.2	9.6
14	43.43	15.8	12.0
15	38.76	16.7	13.8
16	52.83	14.8	12.5
17	42.82	13.9	12.2
18	53.31	12.7	10.3
19	52.7	13.0	10.5
20	49.65	14.8	11.7
21	42.75	17.9	14.3
22	37.74	16.9	14.2
23	59.84	11.5	9.8
24	40.38	15.8	13.2
25	50.87	12.9	11.2
26	55.64	11.4	9.4
27	50.87	14.2	11.2
28	45.19	17.5	13.4
29	34.76	15.9	13.3
30	48.7	12.1	10.0
31	45.56	14.2	11.2
32	45.35	13.5	11.3
33	45.49	17.3	13.3
34	41.6	15.6	12.7
35	37.23	17.0	14.4
36	41.9	15.3	12.6
37	51.89	13.3	10.8
38	52.09	12.8	10.7
39	44.95	15.9	11.4
40	52.09	12.8	10.7

Table 4.6: Readability Score table BERT

4.1.5 COMBINED SCORES

For better comparison of the scores we combined the scores for each of the matrices in tables separately.

Table 4.7: Rouge-1 Score of All 3 Method combined

P no.	Rouge-1 (BERT)			Rouge-1 (LexRank)			Rouge-1 (TextRank)		
	r	p	f	r	p	f	r	p	f
1	0.3168	0.1403	0.1945	0.37623	0.1719	0.236	0.3762	0.1645	0.2289
2	0.4927	0.1387	0.2165	0.5797	0.1532	0.2424	0.5362	0.1316	0.2114
3	0.2527	0.0916	0.1345	0.2857	0.0984	0.1464	0.3186	0.105	0.158
4	0.7058	0.0662	0.1212	0.647	0.0718	0.1294	0.67647	0.07233	0.13068
5	0.2882	0.1481	0.1957	0.4234	0.1747	0.2473	0.42342	0.17029	0.24289
6	0.2631	0.08333	0.1265	0.4107	0.09745	0.15753	0.42105	0.1	0.1616
7	0.5555	0.07692	0.1351	0.5	0.06896	0.12121	0.56756	0.075	0.13249
8	0.3921	0.2409	0.2985	0.4313	0.18965	0.26347	0.42718	0.188	0.26112
9	0.4629	0.1373	0.2118	0.566	0.1287	0.20979	0.5555	0.1287	0.20905

10	0.2328	0.0934	0.1333	0.45205	0.1061	0.17187	0.48648	0.1132	0.1836
11	0.3285	0.1095	0.1642	0.37681	0.115	0.17627	0.3857	0.11297	0.17475
12	0.3	0.2389	0.266	0.33333	0.17143	0.22641	0.34065	0.16756	0.22463
13	0.5	0.08791	0.1495	0.5	0.08121	0.13973	0.48484	0.07511	0.13008
14	0.326	0.0931	0.1449	0.43478	0.09174	0.15152	0.4255	0.08888	0.14705
15	0.4545	0.0847	0.1428	0.45454	0.07317	0.12605	0.44117	0.06944	0.11999
16	0.3411	0.1266	0.1847	0.44705	0.1484	0.22287	0.45348	0.14716	0.22222
17	0.2296	0.1435	0.1766	0.2888	0.2888	0.20259	0.27205	0.14285	0.18734
18	0.3913	0.0511	0.0904	0.43478	0.04878	0.08771	0.43478	0.04716	0.0851
19	0.3571	0.03703	0.06711	0.42857	0.02912	0.05454	0.42857	0.02912	0.05454
20	0.4084	0.0963	0.1559	0.46479	0.12547	0.1976	0.47887	0.12592	0.19941
21	0.279	0.13259	0.17977	0.372	0.1264	0.1887	0.37209	0.12648	0.18879
22	0.30821	0.1557	0.20689	0.3356	0.1731	0.2284	0.32653	0.14906	0.20469
23	0.35849	0.10982	0.16814	0.4528	0.0845	0.14243	0.42592	0.08273	0.13855
24	0.28301	0.06944	0.1115	0.4634	0.0763	0.131	0.45238	0.0745	0.12794
25	0.34146	0.0549	0.09459	0.55555	0.16949	0.2597	0.51388	0.14285	0.22356
26	0.23611	0.1164	0.1559	0.53125	0.08673	0.1491	0.53125	0.08415	0.14529
27	0.15625	0.03937	0.06289	0.36363	0.07881	0.12955	0.36363	0.07881	0.12955
28	0.36363	0.04863	0.0857	0.41666	0.0684	0.11764	0.47916	0.07443	0.12885
29	0.3125	0.05639	0.0955	0.4382	0.13636	0.20799	0.46067	0.12424	0.1957
30	0.2247	0.2409	0.2325	0.582	0.2335	0.3333	0.58208	0.23353	0.33333
31	0.4477	0.1115	0.1785	0.3571	0.1345	0.1954	0.30722	0.21428	0.25247
32	0.1867	0.1448	0.1631	0.31818	0.1359	0.1904	0.38095	0.14285	0.20779
33	0.25	0.1304	0.1714	0.2962	0.0717	0.1155	0.39772	0.15695	0.22508
34	0.29545	0.1585	0.2063	0.34666	0.1097	0.1666	0.42592	0.09704	0.15807
35	0.2777	0.0678	0.10909	0.3787	0.08771	0.1424	0.53333	0.14035	0.22222
36	0.2933	0.1419	0.1913	0.317	0.0575	0.0973	0.46268	0.13716	0.2116
37	0.3066	0.1173	0.1697	0.2903	0.1075	0.1569	0.36585	0.05882	0.10135
38	0.28787	0.1043	0.1532	0.32	0.1016	0.1543	0.59139	0.23504	0.33639
39	0.2682	0.12087	0.16666	0.2941	0.0867	0.1339	0.4	0.17341	0.24193
40	0.468	0.24175	0.3188	0.4153	0.095	0.1547	0.52941	0.07541	0.13202

Table 4.8: Rouge-2 Score of All 3 Method combined

P no.	**Rouge-2(BERT)**			**Rouge-2(LexRank)**			**Rouge-2(TextRank)**		
	r	p	f	r	p	f	r	p	f

1	0.0699	0.0294	0.0414	0.1118	0.04481	0.06399	0.0979	0.0373	0.054
2	0.0568	0.0128	0.0209	0.0909	0.01834	0.0305	0.0681	0.013	0.0218
3	0.0427	0.0115	0.0181	0.04273	0.0107	0.01712	0.0769	0.0183	0.0296
4	0.2	0.0118	0.0224	0.175	0.0123	0.02306	0.175	0.0117	0.02194
5	0.0625	0.0282	0.0389	0.1125	0.0393	0.05825	0.1125	0.038216	0.05705
6	0.0138	0.0037	0.0058	0.05633	0.0102	0.01727	0.05555	0.0101	0.01709
7	0.0975	0.0099	0.01797	0.12195	0.0124	0.0225	0.09523	0.00917	0.01673
8	0.118	0.07112	0.0887	0.15972	0.06149	0.0888	0.15862	0.06117	0.08829
9	0.1492	0.03558	0.0574	0.19696	0.032418	0.05567	0.19402	0.032418	0.05555
10	0.056	0.0213	0.03092	0.08411	0.01768	0.02922	0.11111	0.0229	0.03797
11	0.0947	0.02631	0.0411	0.09574	0.02295	0.03703	0.10526	0.02403	0.03913
12	0.0416	0.03125	0.03571	0.075	0.031578	0.044444	0.07438	0.02941	0.04215
13	0.1666	0.0212	0.0376	0.11111	0.01212	0.02185	0.16216	0.01657	0.03007
14	0.0689	0.0173	0.0277	0.0862	0.014326	0.02457	0.08474	0.01373	0.02364
15	0.125	0.0186	0.0324	0.15	0.017543	0.03141	0.14634	0.01652	0.0297
16	0.1008	0.0298	0.046	0.15966	0.04042	0.064516	0.175	0.04375	0.06999
17	0.021	0.0119	0.0152	0.03157	0.01435	0.01973	0.03141	0.01405	0.01941
18	0.1304	0.011	0.02033	0.13043	0.00872	0.01634	0.13043	0.00824	0.0155
19	0	0	0	0	0	0	0	0	0
20	0.0681	0.01176	0.02	0.05681	0.01131	0.01886	0.07954	0.0158	0.02636
21	0.0593	0.02611	0.0362	0.11016	0.03201	0.0496	0.11016	0.03201	0.04961
22	0.0414	0.01973	0.02674	0.07373	0.03524	0.0476	0.06422	0.02692	0.03794
23	0.076923	0.01838	0.02967	0.13846	0.01898	0.03339	0.13636	0.01931	0.03383
24	0.0615	0.01179	0.0198	0.0652	0.0071	0.01287	0.06382	0.00691	0.01247
25	0.04347	0.00467	0.00843	0.16666	0.0374	0.06109	0.21111	0.04185	0.06985
26	0.04444	0.01877	0.0264	0.03125	0.003508	0.006309	0.03125	0.00335	0.00606
27	0	0	0	0.03703	0.0065	0.01108	0.03703	0.00651	0.01108
28	0.037	0.0035	0.0064	0.01694	0.00202	0.003	0.05084	0.00548	0.0099
29	0.0508	0.0069	0.0122	0.1681	0.04112	0.06608	0.15929	0.03243	0.05389

30	0.0265	0.028	0.02727	0.23913	0.08906	0.12979	0.23913	0.08906	0.12979
31	0.07608	0.01631	0.0268	0.0614	0.0193	0.02941	0.06637	0.03846	0.0487
32	0.0265	0.01807	0.0215	0.025	0.0095	0.01379	0.11403	0.03768	0.05664
33	0.04385	0.0209	0.0283	0.0845	0.0162	0.0272	0.05833	0.01897	0.02862
34	0.01666	0.0085	0.01129	0.07777	0.0179	0.0292	0.21126	0.03856	0.06521
35	0.0845	0.01648	0.02758	0.0864	0.0145	0.0248	0.17777	0.03319	0.05594
36	0.0444	0.0176	0.02531	0.0697	0.0084	0.0149	0.10975	0.02521	0.041
37	0.0333	0.0105	0.01599	0.0422	0.01503	0.0221	0.09302	0.01002	0.01809
38	0.0246	0.0072	0.01126	0.07	0.0186	0.02947	0.3169	0.11688	0.17077
39	0	0	0	0.0327	0.0079	0.0127	0.13	0.05138	0.07365
40	0.2167	0.1131	0.14868	0.0681	0.0119	0.0203	0.13114	0.01335	0.02424

Table 4.9: Rouge-L Score of All 3 Method combined

P no.	**Rouge-L(BERT)**			**Rouge-L(LexRank)**			**Rouge-L(TextRank)**		
	r	p	f	r	p	f	r	p	f
1	0.297	0.1315	0.1823	0.3564	0.1628	0.2236	0.3465	0.1515	0.2108
2	0.4492	0.1265	0.1974	0.5072	0.134	0.2121	0.4927	0.1209	0.1942
3	0.21978	0.0796	0.1169	0.2637	0.0909	0.1352	0.3076	0.1014	0.1525
4	0.6764	0.0635	0.1161	0.6176	0.06862	0.1235	0.67647	0.07232	0.13068
5	0.2252	0.1157	0.1529	0.3603	0.14869	0.2105	0.36036	0.14492	0.20671
6	0.1578	0.05	0.0759	0.3214	0.07627	0.1232	0.33333	0.07916	0.12794
7	0.5277	0.073	0.1283	0.5	0.0689	0.1212	0.5135	0.06785	0.11987
8	0.3431	0.2108	0.2611	0.4215	0.18534	0.2574	0.41747	0.18376	0.25519
9	0.4629	0.1373	0.2118	0.5660	0.12875	0.2097	0.55555	0.12875	0.20905
10	0.2191	0.0879	0.1254	0.3698	0.08681	0.1406	0.43243	0.10062	0.16326
11	0.3	0.1	0.1499	0.3333	0.10176	0.1559	0.34285	0.10041	0.1553
12	0.2555	0.2035	0.2266	0.3111	0.16	0.2113	0.31868	0.15675	0.21014

13	0.5	0.0879	0.1495	0.5	0.08121	0.1397	0.45454	0.07042	0.12195
14	0.326	0.0931	0.1449	0.4347	0.09174	0.1515	0.42553	0.08888	0.14705
15	0.3939	0.07344	0.1238	0.3939	0.06341	0.1092	0.38235	0.06018	0.10399
16	0.2941	0.1091	0.1592	0.4	0.13281	0.1994	0.38372	0.12452	0.18803
17	0.2148	0.1342	0.1652	0.2518	0.136	0.1766	0.23529	0.12355	0.16202
18	0.3913	0.0511	0.0904	0.4347	0.0487	0.0877	0.43478	0.04716	0.08510
19	0.3571	0.03703	0.0671	0.4285	0.02912	0.0545	0.42857	0.02912	0.05454
20	0.3802	0.0897	0.1451	0.4225	0.11406	0.1796	0.39436	0.1037	0.16422
21	0.2325	0.11049	0.1498	0.3372	0.11462	0.1710	0.3372	0.11462	0.17109
22	0.27397	0.13840	0.1839	0.2945	0.1519	0.2004	0.28571	0.13043	0.1791
23	0.30188	0.09248	0.14159	0.4339	0.0809	0.1364	0.38888	0.07553	0.1265
24	0.24528	0.06018	0.09665	0.4634	0.0763	0.1310	0.45238	0.0745	0.12794
25	0.31707	0.05098	0.08783	0.4861	0.1483	0.2272	0.45833	0.12741	0.19939
26	0.23611	0.11643	0.15596	0.5	0.0816	0.1403	0.46875	0.07425	0.1282
27	0.15625	0.03937	0.06289	0.3181	0.0689	0.1133	0.31818	0.06896	0.11336
28	0.25	0.0334	0.0589	0.375	0.06164	0.1058	0.41666	0.06472	0.11204
29	0.2916	0.0526	0.0891	0.4044	0.12587	0.1919	0.4382	0.11818	0.18615
30	0.2022	0.2168	0.2093	0.5522	0.2215	0.3162	0.55223	0.22155	0.31623
31	0.4179	0.104	0.1666	0.3095	0.1165	0.1693	0.28915	0.20168	0.23762
32	0.1686	0.1308	0.1473	0.2613	0.1116	0.1564	0.35714	0.13392	0.1948
33	0.2142	0.1118	0.1469	0.2777	0.0672	0.1083	0.36363	0.14349	0.20578
34	0.2613	0.1402	0.1825	0.32	0.1012	0.1538	0.35185	0.08016	0.13058
35	0.2592	0.0633	0.10181	0.3181	0.0736	0.1196	0.53333	0.14035	0.22222
36	0.28	0.13548	0.1826	0.2926	0.053	0.0898	0.43283	0.12831	0.19795
37	0.28	0.1071	0.1549	0.258	0.0956	0.1395	0.34146	0.0549	0.09459
38	0.2575	0.0934	0.137	0.2666	0.0847	0.1286	0.55913	0.22222	0.31804
39	0.2439	0.10989	0.1515	0.2941	0.0867	0.1339	0.38666	0.16763	0.23387
40	0.4255	0.21978	0.28985	0.3692	0.0845	0.1375	0.49019	0.06983	0.12224

Table 4.10: Readability Score of All 3 Method combined

P. No	FLESCH READING EASE			INDEX			FLESCH KINCAID GRADE		
	LexRank	BERT	TextRank	LexRank	BERT	TextRank	LexRank	BERT	TextRank
1	43.83	45.76	42.41	13.3	14.5	14	11.8	11.1	12.4
2	49.25	42	48.23	14.4	15.1	14.8	11.8	12.5	12.2
3	44.78	44.07	43.36	16.5	17.3	17.1	13.5	13.8	14.1
4	41.53	46.71	39.71	17.2	16	18	14.8	12.8	15.5
5	53.95	56.79	53.55	14.2	13.9	14.2	12.1	11	12.3
6	48.54	44.44	48.54	13.5	12.8	13.5	12.1	11.6	12.1
7	41.19	41.4	40.48	15.4	16.2	15.4	12.9	12.8	13.1
8	42	40.28	42	14.4	17.6	14.4	12.5	13.2	12.5
9	44.37	36.02	44.37	17.1	19.3	17.1	13.7	14.8	13.7
10	36.63	45.69	35.91	16.7	16.8	17	14.6	13.2	14.9
11	41.5	39.26	39.97	15.1	17	15.9	12.7	13.6	13.3
12	59.03	60.55	59.03	12.8	13.1	12.8	10.1	9.6	10.1
13	54.32	55.13	54.32	10.9	11.2	10.2	9.9	9.6	9.9
14	52.9	43.43	52.09	12.4	15.8	13.3	10.4	12	10.7
15	43.97	38.76	43.97	16.5	16.7	16.5	13.9	13.8	13.9
16	53.75	52.83	53.04	13.5	14.8	13.9	12.2	12.5	12.4
17	41.7	42.82	41.09	13.7	13.9	14.1	12.7	12.2	12.9
18	54.02	53.31	53.71	12	12.7	12	10	10.3	10.1
19	50.36	52.7	50.36	13.4	13	13.4	11.4	10.5	11.4
20	55.17	49.65	56.89	15.1	14.8	14.1	11.6	11.7	11
21	44.17	42.75	44.17	17.1	17.9	17.1	13.8	14.3	13.8
22	46.61	37.74	43.16	15.1	16.9	17.5	12.8	14.2	14.2

23	56.18	59.84	56.39	13	11.5	12.9	11.2	9.8	11.2
24	39.87	40.38	47.42	14.2	15.8	14.5	13.4	13.2	12.5
25	52.19	50.87	49.25	11.9	12.9	12.9	10.7	11.2	11.8
26	56.25	55.64	55.64	10.1	11.4	10.9	9.1	9.4	9.4
27	52.6	50.87	52.6	13.2	14.2	13.2	10.5	11.2	10.5
28	46.1	45.19	39.5	16.3	17.5	19.5	13	13.4	15.6
29	40.58	34.76	36.63	14.5	15.9	16.6	13.1	13.3	14.6
30	46.47	48.7	46.47	12.3	12.1	12.3	10.8	10	10.8
31	51.28	45.56	42	13.2	14.2	14.1	11.1	11.2	12.5
32	52.9	45.35	51.07	12.6	13.5	13.7	10.4	11.3	11.1
33	40.08	45.49	40.08	14.4	17.3	14.4	13.3	13.3	13.3
34	41.6	41.6	41.6	14.2	15.6	14.2	12.7	12.7	12.7
35	35.1	37.23	35.1	16.8	17	16.8	15.2	14.4	15.2
36	40.79	41.9	40.79	15	15.3	15	13	12.6	13
37	51.18	51.89	51.28	13	13.3	12.8	11.1	10.8	11.1
38	51.99	52.09	42.92	12	12.8	12.5	10.8	10.7	12.2
39	41.29	44.95	41.29	14.9	15.9	14.9	12.8	11.4	12.8
40	53.65	52.09	38.49	15	12.8	19.1	12.2	10.7	16

4.1.6 AVERAGE SCORES

After getting all the results from the models we calculated the average of the score for the 3 models. Results are:

Method	Rouge-1			Rouge-2			Rouge-L			Readability		
	r	p	f	r	p	f	r	p	f	Reading ease	Index	Kincaid grade
BERT	0.3386	0.1162	0.1632	0.06728	0.0195	**0.09209**	0.30765	0.1045	0.1471	46.41	**14.907**	12.04
LexRank	0.417	0.1163	0.1726	0.0928	0.0211	0.0332	0.3931	0.1029	0.1573	**47.34**	14.1 72	12.14 25
TextRank	**0.44590 825**	**0.1214 219**	**0.18425 025**	**0.11614 925**	**0.0270 976**	0.04233 125	**0.41146 175**	**0.11171 293**	**0.1695 059**	45.97 2	14.6 65	**12.57**

Table 4.11: Average scores

4.1.7 GRAPHICAL REPRESENTATION

After getting the average values we plot them in graphical form for better visualisation of the comparison between the three methods.

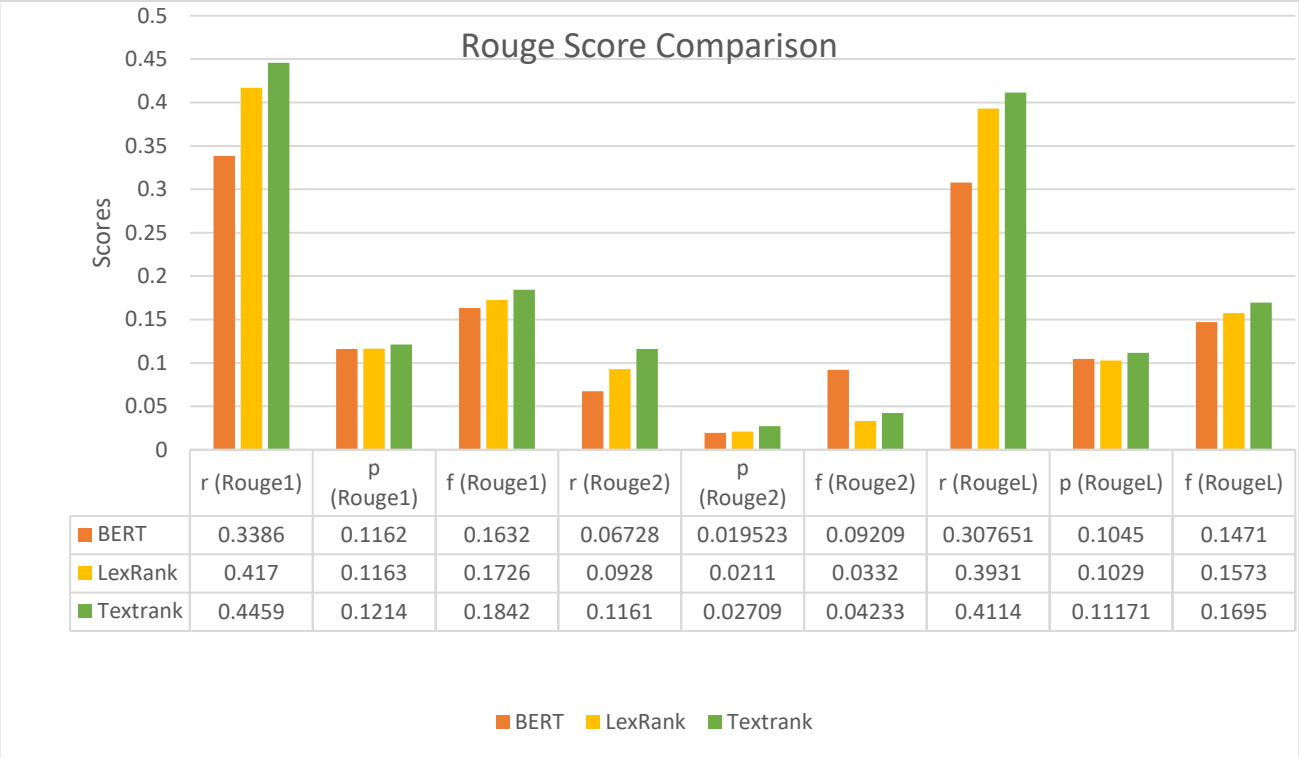
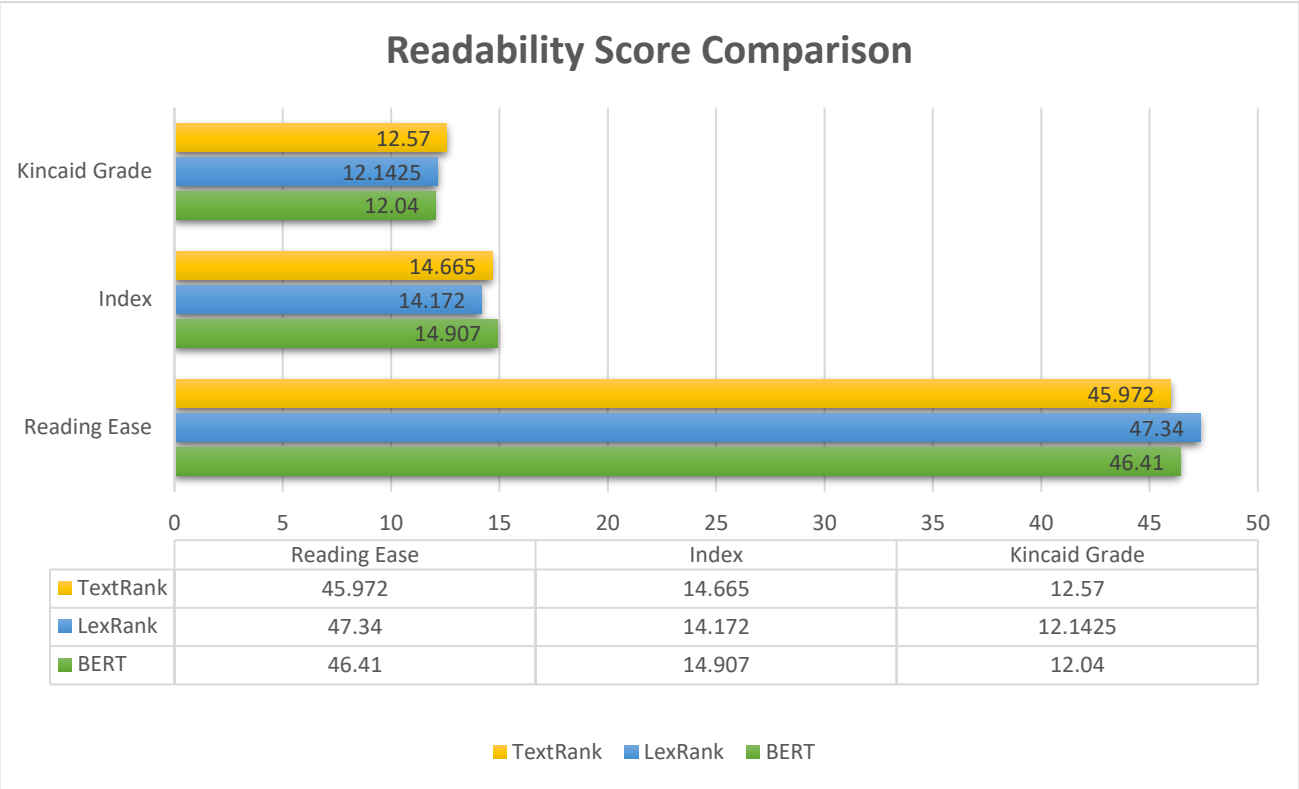
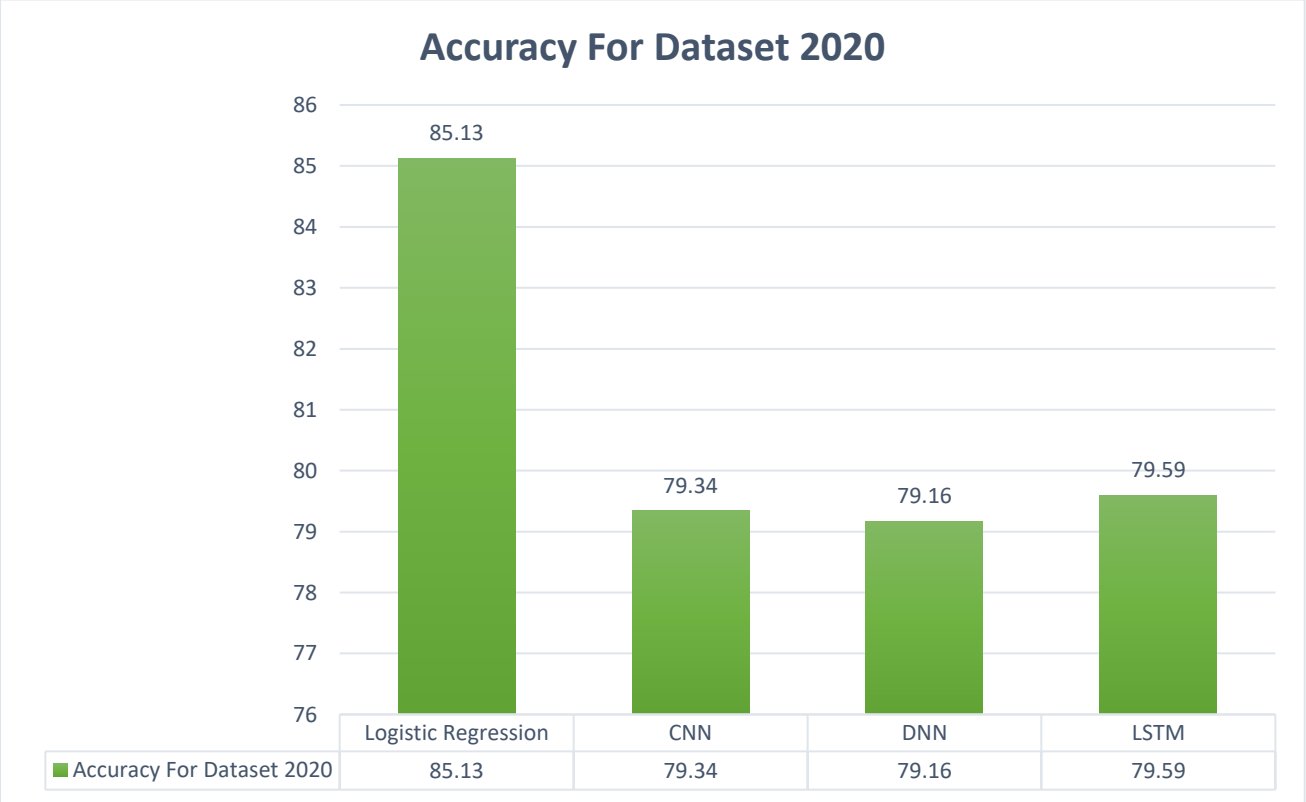


Fig 4.2: Representation of ROUGE score comparison



	Reading Ease	Index	Kincaid Grade
TextRank	45.972	14.665	12.57
LexRank	47.34	14.172	12.1425
BERT	46.41	14.907	12.04

Fig 4.3: Representation of READABILITY Score comparison



	Logistic Regression	CNN	DNN	LSTM
Accuracy For Dataset 2020	85.13	79.34	79.16	79.59

Fig 4.4: Accuracy Comparison for each model for dataset 2020

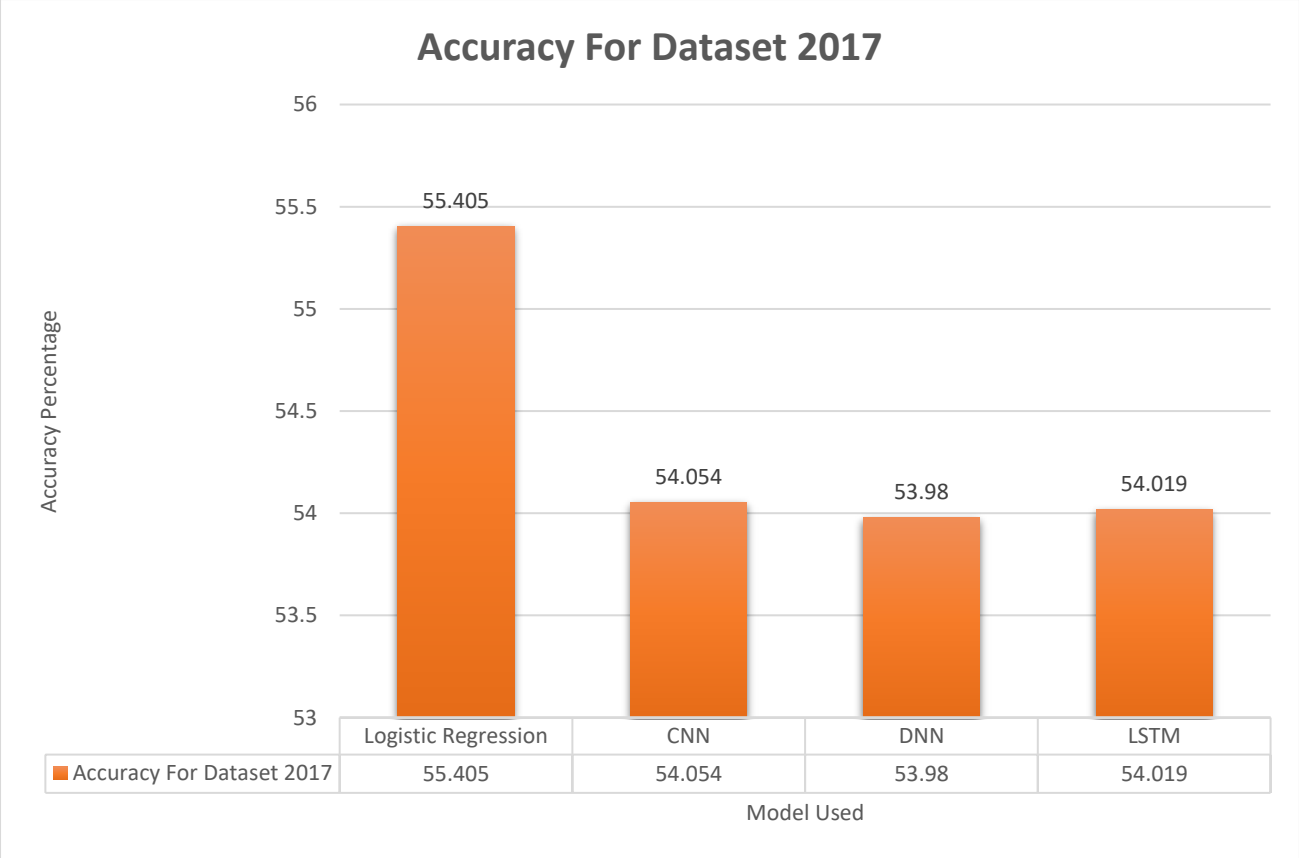


Fig 4.5: Accuracy Comparison for each model for dataset 2017

CHAPTER 5

5.1 CONCLUSION

This experimental study has been conducted to find out the best suited algorithm for the purpose of academic reviews summarization of a given dataset. This leads us to the conclusion that the TextRank algorithm provides better results in terms of ROUGE score than the LexRank and the BERT algorithm. This is not to discredit other algorithms; they don't seem to work quite as good on some types of data.

Also, for Kincaid Grade Level, TextRank performed better than the other two. For Readability Index, BERT performed well and for Reading ease LexRank performed well. We achieved what we wanted to do with the study and we have come to a conclusion that specific algorithms work good for a particular type of data.

As TextRank is working relatively well in case of Academic reviews as concluded, we will use TextRank algorithm for our future operations.

In conclusion, our research efforts focused on the automation of text summarization and the compilation of meta-reviews. We were able to develop and deploy a system capable of effectively summarising textual content and generating comprehensive meta-reviews by employing sophisticated techniques.

Through meticulous experimentation and evaluation, we conducted comparative analyses of several deep learning models, including convolutional neural networks (CNN), deep neural networks (DNN), long short-term memory (LSTM), and logistic regression. In terms of accuracy, our findings demonstrated unequivocally that logistic regression outperformed the aforementioned deep learning models.

This significant result demonstrates that logistic regression is an effective method for text summarization and meta-review generation. Its robust predictive capabilities and inherent characteristics enable it to effectively capture and comprehend the underlying structure and semantics of textual data, resulting in superior accuracy in producing accurate summaries and thorough meta-reviews.

Our study has provided empirical evidence that not only supports the superiority of logistic regression

but also contributes to the expanding corpus of knowledge in the fields of natural language processing and machine learning. This analysis of deep learning models in the context of text summarization and meta-review generation provides valuable insights for researchers, practitioners, and developers seeking to improve the efficiency and effectiveness of automated text analysis systems.

5.2 FUTURE WORK

There are still unexplored options for enhancing and expanding our research work. This section outlines two promising future research directions: predicting the best paper using meta-reviews and implementing an on-desk rejection system for scholarly articles. First, using meta-reviews to forecast the best paper has the potential to improve the efficacy and quality of the peer review process. By analysing the aggregated opinions and ratings of reviewers, we can create a machine learning model that predicts which papers are most likely to receive high praise and recognition. This prediction model can aid conference organisers and journal editors in the early identification of exceptional papers, enabling them to allocate resources more efficiently and prioritise the most influential research. In addition, such a system can assist researchers and authors in gaining insight into the strengths and shortcomings of their work, thereby facilitating targeted enhancements and enhancing their likelihood of success. Implementing an on-desk rejection system for scholarly articles can greatly accelerate the publication process. By utilising techniques for natural language processing, we can create an application that autonomously evaluates submitted manuscripts and identifies those that do not meet the required quality or relevance standards. By promptly rejecting papers that are unlikely to be accepted, such a system can save valuable time for both authors and reviewers. In addition, it can provide authors with detailed feedback, highlighting the specific aspects that require development prior to resubmission, thereby nurturing a more efficient and iterative publication cycle.

Bibliography

[1] Bhatia, C., Pradhan, T., & Pal, S. (2020, July). MetaGen: an academic meta-review generation system. In Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (pp. 1653-1656).

[2] Shen, C., Cheng, L., Zhou, R., Bing, L., You, Y., & Si, L. (2022). MReD: A meta-review dataset for structure-controllable text generation. Findings of the Association for Computational Linguistics: ACL 2022, 2521-2535.

[3] Dongyeop Kang, Waleed Ammar, Bhavana Dalvi Mishra, Madeleine van Zuylen, Sebastian Kohlmeier, Eduard Hovy, Roy Schwartz. (2018). A Dataset of Peer Reviews (PeerRead): Collection, Insights and NLP Applications. *Review Aspect Score Prediction.*