Text Mining and Search Project

Text Classification and Text Summarization on Web of Science Dataset

Arizzi Sara 845374, Bidone Federico 892054

A.Y. 2023/2024

Abstract

This report navigates the application of advanced natural language processing techniques in a domain-specific context, focusing on text classification and summarization tasks within the Web of Science dataset. For the text classification task we harnessed the power of BERT, or Bidirectional Encoder Representations from Transformers, a state-of-the-art transformer model. Using its pre-trained contextual embeddings, we aimed to categorize documents effectively, particularly within the domain-specific context of the Web of Science dataset. Then we delve into extractive summarization, utilizing two distinct textual representations: GloVe embeddings for semantic context and tf-idf vectors for term frequency analysis. This approach aims to distill essential information from lengthy documents, providing concise and informative summaries. Finally for abstractive summarization, we employ BART, a powerful transformer-based model. We both fine-tuned it on our training data to tailor its understanding to our specific domain, but we also used the pre-trained DistilBART model from Hugging Face. Our findings not only showcase the efficacy of BERT for text classification but also underscore the strengths of different methods in extractive and abstractive summarization.

Contents

1 Introduction

In the ever-expanding landscape of textual data, the ability to sift through vast amounts of information and extract meaningful insights is a crucial skill. Text classification and text summarization emerge as powerful techniques in the realm of natural language processing, offering distinct solutions to the challenges posed by extensive textual datasets.

Text classification, a cornerstone of natural language processing, involves the categorization of textual data into predefined classes or categories. Whether discerning sentiment from customer reviews, classifying news articles by topic, or identifying spam emails, text classification lays the foundation for various applications. Throughout this report, we will explore the intricacies of text classification, from preprocessing raw text to training machine learning models. Leveraging a dataset from the Web of Science (WOS), we showcase practical examples and elucidate the decision-making process involved in building a robust text classification system.

Text summarization addresses the challenge of distilling lengthy documents into concise, information-rich summaries. As the volume of textual information continues to surge, the need for automated summarization becomes increasingly apparent. Whether condensing news articles, academic papers, or user-generated content, text summarization algorithms offer a means to extract key insights efficiently. Throughout the report, we guide the reader through the intricacies of abstractive and extractive summarization techniques, shedding light on the nuances of generating coherent and contextually relevant summaries.

1.1 Dataset

The **Web of Science (WOS)** is a dataset for document classification introduced by Kowsari et al. in their work on HDLTex: Hierarchical Deep Learning for Text Classification[1]. It contains 46,985 documents and these documents are classified into 134 sub-domains or areas of the paper categories with 7 domain categories. The dataset has 3 different variants:

1. **WOS-46985** is the full dataset that contains 46,985 documents with 7 major domains and 134 sub-domains;
2. **WOS-11967** is a sub-sample dataset, that contains 11,967 documents with 7 major domains and 135 sub-domains;
3. **WOS-5736** is a sub-sample dataset, that contains 5,736 documents with 3 major domains and 11 sub-domains.

For our text classification and summarization project, we employed different variants of the Web of Science (WOS) dataset to accommodate varying computational resource limitations. Our text classification tasks were conducted using the WOS-46985 dataset. Instead, for text summarization, we utilized the WOS-5736 dataset, which consists of 5,736 documents. The decision to work with this smaller dataset was influenced by our restricted computational capacity.

The documents in the dataset are organized into seven distinct classes, each representing a specific domain category. The classes are denoted by labels ranging from Class 0 to Class 6.

The WOS-46985 dataset contains all seven domain labels: Computer Science (Class 0), Electrical Engineering (Class 1), Psychology (Class 2), Mechanical Engineering (Class 3), Civil Engineering (Class 4), Medical Science (Class 5), Biochemistry (Class 6). In contrast, the WOS-5736 dataset focuses on a more limited subset, containing only three domain labels: Biochemistry, Electrical Engineering, and Psychology.

1.2 Ground Truth Acquisition

The dataset used for the text summarization task is WOS-5736 dataset, which has been acquired directly from an online source through a URL. Given that the Web of Science dataset is primarily designed for text classification, the absence of ground truth summaries posed a challenge for our text summarization task. Thus, to generate ground truth summaries we explored various alternatives, including Cohere APIs and OpenAI APIs. However, due to limitations in the number of requests allowed on a free subscription, we opted for Selenium automation on the QuillBot Summarization website. Despite its effectiveness, it's noteworthy that the process of generating 5736 summaries consumed approximately 15 hours (roughly 10s per sample), highlighting its relative inefficiency. Additionally, the QuillBot website's implementation of bot detection mechanisms necessitated deliberate pacing of the automation code to avoid detection, contributing to the overall time investment in the summarization process. In 1 we compare WOS-5736 with the generated ground truth summaries.

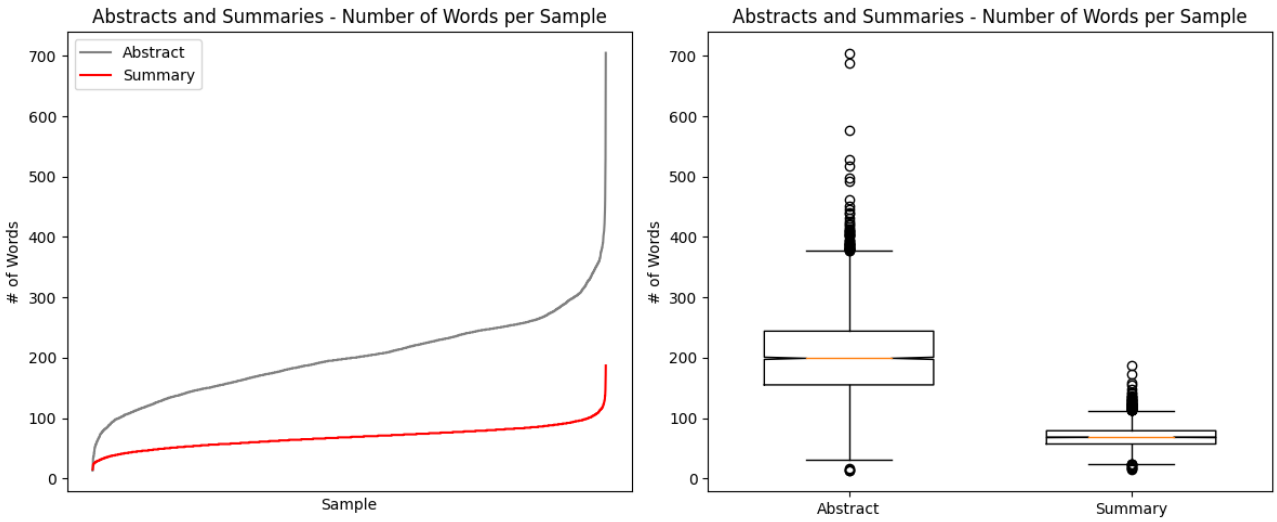


Figure 1: Comparison between original abstracts and QuillBot summaries

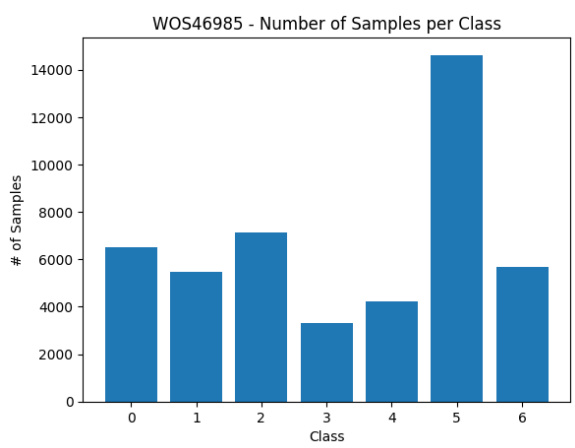
2 Exploratory Data Analysis

Exploratory data analysis is a fundamental phase in every data science project. It allows to understand the structure and characteristics of the data. In the case of the presented project, the dataset WOS-46985 contains 46,985 samples divided into 7 unique classes, and the dataset WOS-5736 contains 5,736 samples divided into 3 unique classes. These information are crucial to understand the complexity of the classification problem.

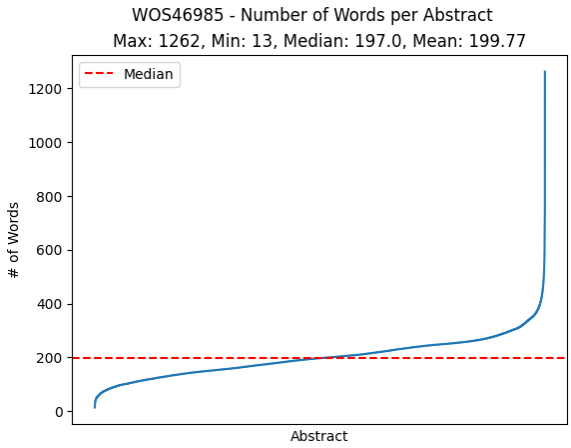
2.1 WOS-46985

A bar chart was created (Figure 2a) to visualize the distribution of samples among the different classes. This type of visualization helps to understand if the classes are balanced or if some classes are over-represented. In the specific case, class 5 has significantly more samples than the others.

Finally, a line and box plot were created (Figure 2b) to visualize the distribution of the number of words per sample. This analysis reveals that most of the samples have a word count in the range 100-400. In particular, the

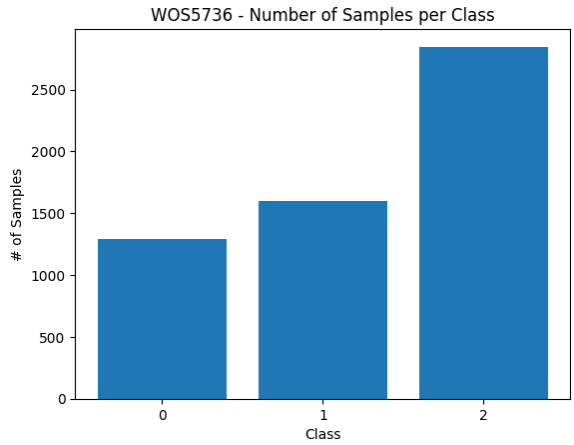


(a)

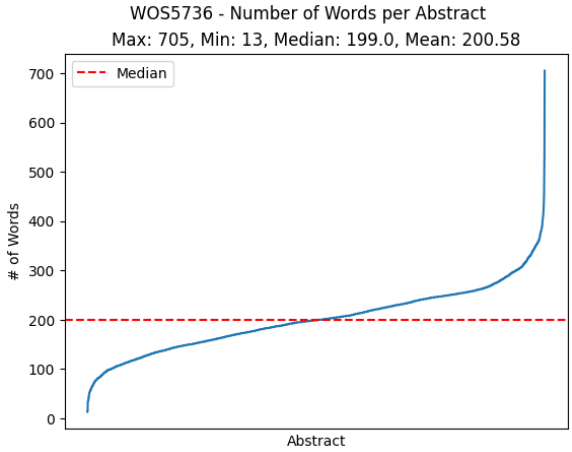


(b)

Figure 2: in Figure 2a a Bar chart is showing the number of samples per class. Class 5 has the highest number of samples, close to 14000. Classes 1, 2, and 6 have similar but lower counts, ranging between approximately 6000 and 8000 samples. Class 3 has the lowest count. In Figure 2b a line chart is showing the distribution of the number of words per sample and the median value.



(a)



(b)

Figure 3: in Figure 3a a Bar chart showing the number of samples per class. Class 2 has the highest number of samples, more than 2500. Classes 0 and 1 have lower counts, approximately 1300 and 1600 samples. Class 0 has the lowest count. In Figure 3b a line chart is showing the distribution of the number of words per sample and the median value.

samples lenght goes from a minimum of 13 to a maximum of 1262. The median value is 197. This information can be useful to set the maximum input length for the text classification model.

2.2 WOS-5736

A bar chart was created (Figure 3a) to visualize the distribution of samples among the different classes. This type of visualization helps to understand if the domain are balanced or if some domains are over-represented. In the specific case, class 2 has more samples than the others.

Finally, a line and box plot were created (Figure 3b) to visualize the distribution of the number of words per sample. This analysis reveals that most of the samples have a word count in the range 100-350. In particular, the samples lenght goes from a minimum of 13 to a maximum of 705. The median value is 199. This information can be useful to set the maximum and minimum summary length for the text summarization model.

3 Text Classification

3.1 Data Acquisition and Pre-processing

Data acquisition takes place directly from an online source through a URL. This method of data acquisition is very common when working with large publicly available datasets. The downloaded file is a zip file, a data compression format that allows to reduce the necessary storage space and facilitates data transfer. Once downloaded, the zip file is extracted in the current working directory.

Subsequently, the data is read and the labels are matched using the pandas library [2]. Pandas is a Python library that provides flexible data structures and allows to easily work with structured data. In this case, it is used to load an Excel file containing the metadata. During this phase, the labels are cleaned to ensure consistency.

Finally, the text data for training is read from a text file. This data is accompanied by subdomain labels, domain labels, and child labels. This pre-processing phase is crucial to prepare the data in a suitable format for model training.

3.2 Data Preparation and Dataset Class Definition

To prepare the data for model training, only the 'Abstract' and 'Y1' columns were selected from the dataframe. This choice is guided by the fact that the goal is to classify the abstracts based on the 'Y1' labels.

Subsequently, an AbstractsDataset class was defined that inherits from torch.utils.data.Dataset. This custom class is used to prepare the data for training and testing. The definition of a custom dataset class is a common practice when working with PyTorch [3], a popular deep learning framework. This class allows to organize the data in a format that can be easily used by PyTorch for model training.

Finally, the data is prepared for training. This process includes encoding the abstracts using a pre-trained BERT tokenizer and splitting the dataframe into training and test sets (80/20). The BERT[4] tokenizer is used to convert the text into a format that can be understood by the BERT model. Splitting the data into training and test sets is a fundamental step to evaluate the performance of the model.

3.3 Model Creation and Training

The model used for this project is BertForSequenceClassification, a pre-trained BERT model adapted for sequence classification. BERT, which stands for Bidirectional Encoder Representations from Transformers, is a deep learning model designed for natural language processing. It is known for its ability to understand the context of words by looking both to the left and right of each word.

The model is moved to the GPU, if available, otherwise the CPU is used. This choice is guided by the fact that training deep learning models is a computationally intensive process that can greatly benefit from GPU acceleration.

The architecture of the BERT model consists of an embeddings module, an encoder, and a pooler. The embeddings are made up of word embeddings, position embeddings, and token type embeddings. These are summed and normalized to create the input embeddings. The encoder is composed of 12 transformer layers, each of which has an attention module followed by an intermediate module and an output module. The pooler takes the output of the last layer of the encoder and transforms it into a fixed-size vector that can be used for classification. Finally, there is a dropout layer and a linear layer for classification.

For model training, some parameters are set. The number of epochs is set to 5. Epochs represent the number of times the entire dataset is passed forward and backward through the neural network. The optimizer used is AdamW[5] with a learning rate of 1e-5. AdamW is a variant of the Adam optimizer that has better convergence and more stable training. The learning rate determines how fast the model learns; a rate too high can cause instability, while a rate too low can cause slow learning.

The training of the model takes place in a training loop. In each epoch, for each batch of data, the loss is calculated and backpropagated through the model. The loss is a measure of how much the model's predictions deviate from the actual data. Backpropagation is the process of adjusting the weights of the model based on the calculated loss. The technique of gradient clipping is used to prevent gradient explosion, a problem where gradients become too large and cause instability in training. At the end of each epoch, the average loss is printed.

3.4 Model Evaluation

After training, the model is evaluated on the test set. This process includes using the model to make predictions on the test set and calculating the precision, recall, and f1-score metrics for each class. These metrics provide a quantitative measure of the model's performance. Precision is the percentage of correct predictions among all

positive predictions, recall is the percentage of actual positives that were correctly identified, and the f1-score is a harmonic mean of precision and recall.

In addition, an AbstractsDataset is created for the entire dataframe and a corresponding DataLoader. The model is then used to make predictions on the entire dataset and the predictions are compared with the original labels. This step provides a complete evaluation of the model's performance on all available data.

3.5 Training and Evaluation Results

During training, the loss is monitored for each epoch. This information is useful to understand how the model improves over time. In the case of the presented project, the loss consistently decreases from 0.55 to 0.13 over the course of 5 epochs. This indicates that the model is learning from the data and that its performance improves with each epoch. The decrease in loss is a positive signal and suggests that the model is converging towards a solution.

After training, the model is evaluated on the test set and on the entire data set. The evaluation metrics include accuracy, precision, recall, and f1-score. In the case of the presented project, the model achieves an accuracy of 91% on the test set and 97% on the entire data set. These results indicate that the model is able to generalize well on new data and suggest that BERT is a powerful model for text classification.

Table 1: Train-Test metrics

	Precision	Recall	F1-Score	Support
Class 0	0.95	0.94	0.94	1328
Class 1	0.94	0.95	0.94	1072
Class 2	0.86	0.90	0.88	1387
Class 3	0.92	0.88	0.90	662
Class 4	0.91	0.91	0.91	853
Class 5	0.92	0.89	0.90	2918
Class 6	0.86	0.92	0.89	1177
Accuracy			0.91	9397
Macro avg	0.91	0.91	0.91	9397
Weighted avg	0.91	0.91	0.91	9397

Table 2: Total dataset metrics

	Precision	Recall	F1-Score	Support
Class 0	0.98	0.98	0.98	6514
Class 1	0.98	0.99	0.98	5483
Class 2	0.95	0.97	0.96	7142
Class 3	0.98	0.96	0.97	3297
Class 4	0.97	0.97	0.97	4237
Class 5	0.98	0.95	0.96	14625
Class 6	0.93	0.98	0.95	5687
Accuracy			0.97	46985
Macro avg	0.97	0.97	0.97	46985
Weighted avg	0.97	0.97	0.97	46985

As shown in Tables 1 and 2, the over-representation or under-representation of classes was not decisive. Class 3 and Class 5 exhibit similar values, despite one being the smallest and the other the largest. Neither of them have the worst or the best performance. Moreover, the results on the complete dataset are more homogeneous across the classes.

The project has demonstrated the effectiveness of using BERT for text classification. The model achieved an accuracy of 91% on the test set and 97% on the entire data set. These results indicate that the model is able to generalize well on new data. Possible future improvements could include optimizing the model parameters and exploring other text classification models.

4 Text Summarization

Text summarization is the technique which automatically creates an abstract or summary of a text. It gained widespread interest due to overwhelming amount of textual information available in electronic format. Text summarization techniques can be broadly grouped into abstractive summarization (AS) and extractive summarization (ES). Most research on text summarization are ES since it is easier and faster than AS. ES extracts verbatim most salient sentences from text. Meanwhile, AS is relied on Natural Language Processing techniques to copy-paste sentence fragments from the input document and maybe combine the selected content with extra linguistic information in order to generate the final summary. There are two main problems with ES. First, the textual coherence is not guaranteed as resolving anaphora resolution is not paid attention in this approach. Second, redundant phrases still exist in the summary. AS can solve this problem by carrying out NLP techniques to post-process the output of ES such as sentence truncation, aggregation, generalization, reference adjustment and rewording [6].

4.1 Extractive Text Summarization

Extractive text summarization is a technique in natural language processing (NLP) that involves the selection and extraction of key sentences or phrases from a given document to create a concise summary. It identifies and pulls out existing sentences deemed most representative of the document's content. Extractive summarization has the advantage of preserving the wording and style of the source document since it relies on sentences already present in the text. However, it may face challenges in generating concise and coherent summaries, especially for complex or lengthy documents.

4.1.1 Data Preparation

To perform Extractive Summarization we used only the test set, because there is no need to split since we do not have training and testing phases. Starting from raw text, we removed every unnecessary characters and we obtained cleaned abstracts. Then with tokenization, each abstract is segmented into individual sentences (using nltk pre-built tokenization function). From the tokenized sentences we remove special characters, punctuation and stopwords.

Then we use a vectorization technique to obtain a higher representation of each sentence in the abstract. In particular we used two different vectorization techniques:

- **TF-IDF vectorization**, which is a technique used to represent documents as numerical vectors based on the importance of terms within the documents. The importance of terms in the document is measured with Term Frequency (TF) and Inverse Document Frequency (IDF). TF measures how often a term appears in a document and it is calculated as the ratio of the number of occurrences of a term to the total number of terms in the document. IDF measures the importance of a term across a collection of documents. It is calculated as the logarithm of the ratio of the total number of documents to the number of documents containing the term. The TF-IDF score for a term t in a document d is the product of its TF and IDF values. Each document is represented as a vector where each element corresponds to the TF-IDF score of a term. The entire collection of documents can be represented as a matrix, where each row corresponds to a document and each column corresponds to a unique term.
- **GloVe embeddings**, which are dense vector representations of words that capture semantic relationships between words based on their co-occurrence in a given corpus of text. GloVe (Global Vectors for Word Representation) is designed to produce embeddings that reflect global patterns of word co-occurrences, capturing both syntactic and semantic information. In our case, we used the pre-trained 100-dimensional GloVe embedding.

To perform TF-IDF vectorization we used TfidfVectorizer from scikit-learn, which generates a document-term matrix that encapsulates the importance of terms within each sentence. By multiplying this matrix with its transpose, we obtain the similarity matrix related to the abstract. Given this matrix, we then create a similarity graph, where each sentence represent a node and each arc represents the similarity between two sentences.

Instead, to perform vectorization with GloVe embeddings we iterate through the words in each sentence, to retrieve their GloVe embeddings, and then we calculate the average to form the sentence vector. Then for each sentence in an abstract, we create a similarity matrix by calculating the pairwise cosine similarity between the GloVe sentence vectors. Given this matrix, we then create a similarity graph, where each sentence represent a node and each arc represents the cosine similarity between two sentences.

4.1.2 Algorithm

Finally, in both cases, we extract the sentence importance through the application of **PageRank** scores to the similarity graph, constructed using the NetworkX library [7]. The resulting scores show the significance of each

sentence based on its similarity to others. Finally we rank the sentences from highest to lowest, and the top 4 sentences are selected for inclusion in the final summary. To obtain the predicted summary, we concatenate the chosen sentences.

PageRank computes a ranking of the nodes in the graph G based on the structure of the incoming links. It was originally designed as an algorithm to rank web pages.

4.2 Abstractive Text Summarization

Abstractive text summarization is a natural language processing task where systems generate concise summaries by creating new sentences that capture the main ideas of a document. Unlike extractive summarization, which selects and reuses existing sentences, abstractive summarization involves forming a conceptual understanding of the document, generating novel content, and organizing it into a coherent summary. The goal is to produce informative and fluently written summaries that may deviate from the original wording. This approach is valuable when a more condensed or expressive representation is needed.

4.2.1 Data Preparation

The dataset has been divided into two subsets: a training set and a test set, each serving distinct purposes in model development and evaluation. The training set, constituting 80% of the data, was utilized to train the model in the fine-tuning case, allowing it to learn underlying patterns and features. On the other hand, the testing set, representing the remaining 20%, was reserved to assess the model's generalization performance on previously unseen examples.

For the fine-tuned BART model, we used BartSeq2SeqLMPreprocessor, a pre-built preprocessor from keras_nlp that takes care of all the preprocessing steps. Moreover, for the pre-trained BART model we did not perform data preparation since there was no training step.

4.2.2 Model

The model we used is **BART** (Bidirectional and Auto-Regressive Transformers), a sequence-to-sequence model introduced by Mike Lewis et al. in their work BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension [8]. It is designed for a wide range of natural language processing tasks, including text summarization. It employs a bidirectional approach during pre-training, enabling it to consider both left-to-right and right-to-left contexts. This bidirectional understanding enhances the model's ability to capture nuanced relationships and dependencies within the input text. Also during pre-training, BART utilizes a masked language model objective, where certain tokens in the input sequence are masked, and the model is trained to predict those masked tokens. This objective enhances the model's understanding of contextual relationships and helps it generate coherent and contextually appropriate summaries.

Firstly we used BART fine-tuned on our data, to allow the model to better understand and represent the vocabulary specific to the domain. Since we used the version trained on BookCorpus, English Wikipedia and CommonCrawl this step is particularly important, because the terminology used in our dataset differs from that in the original pre-training corpus. Given our limited resources, both in time and in computational resources, we only trained the model on 10 epochs. Which lead to insufficient results.

So we used DistilBART pre-trained model, downloaded from Hugging Face. This model has 306 millions parameters, and its inference time is roughly 307ms per sample. Using this kind of model, allowed us to achieve much better results. Showing that BART is a suitable model for domain-specific text summarization tasks.

4.3 Evaluation Results

To evaluate the predicted summaries, both for extractive and abstractive text summarization, we used **ROUGE** (Recall-Oriented Understudy for Gisting Evaluation) [9]. It includes measures to automatically determine the quality of a summary by comparing it to other (ideal) summaries created by humans. The measures count the number of overlapping units such as n-gram, word sequences, and word pairs between the computer-generated summary to be evaluated and the ideal summaries created by humans. In our case the summaries are produced by an Artificial Intelligence tool, since human generated summaries were not available in the original dataset.

In particular we used ROUGE-1, ROUGE-2, ROUGE-L. ROUGE-1 refers to the overlap of unigrams between the generated and reference summaries. ROUGE-2 refers to the overlap of bigrams between the generated and reference summaries. And ROUGE-L is based on the Longest Common Subsequence, which takes into account sentence-level structure similarity and identifies longest co-occurring in sequence n-grams automatically.

In the evaluation of generated summaries using ROUGE, as we can see from Table 3, there is a notable observation: when applied to extractive summarization tasks, the obtained results tend to be significantly higher compared to abstractive summarization. This discrepancy in performance can be attributed to ROUGE's methodology, which primarily relies on assessing the overlap of n-grams (contiguous sequences of n words)

Table 3: Results Evaluation with ROUGE

Generation Method	ROUGE-1	ROUGE-2	ROUGE-L	Average
PageRank with TF-IDF	0.45	0.24	0.42	0.37
PageRank with GloVe	0.45	0.24	0.43	0.37
Fine-Tuned BART	0.07	0.005	0.06	0.04
Pre-Trained BART	0.40	0.20	0.37	0.32

between the generated and reference summaries. Extractive summarization methods, which select and concatenate existing sentences from the source text, naturally generate summaries that share extensive n-gram overlap with the reference summaries. As a result, ROUGE metrics, particularly those based on precision and recall of n-grams, tend to yield elevated scores for extractive approaches. On the other hand, abstractive summarization involves generating novel sentences, potentially introducing variations in wording and structure. Consequently, the n-gram overlap between the generated and reference summaries may be lower, leading to comparatively lower ROUGE scores. While this does not necessarily reflect a deficiency in the abstractive summarization quality, it underscores the sensitivity of ROUGE to exact n-gram matches. The observed lower ROUGE scores for the fine-tuned model on our limited dataset should be contextualized as a result of resource constraints, preventing extensive training with a larger dataset and longer epochs. Despite these limitations, the model remains well-suited for the summarization task, as evidenced by the robust performance of its pre-trained version. The pre-trained model demonstrates notable success, indicating that the architecture is inherently effective for the summarization task.

5 Conclusion

The project achieved two significant outcomes. Firstly, employing BERT for domain-specific text classification proved to be highly successful. The BertForSequenceClassification model exhibited remarkable accuracy, reaching 91% on the test set and an impressive 97% on the entire dataset. This underscores the effectiveness of BERT in handling nuanced classifications within specific domains. Additionally, the project explored text summarization, covering both extractive and abstractive approaches. Moreover, during evaluation, the use of ROUGE for metric assessment introduced a bias that favored extractive summarization performance. Despite this bias, the abstractive summarization pre-trained model BART achieved comparable performance to extractive summarization. Thus, the reported high performance of BART in text summarization is indicative of its intrinsic capabilities to produce accurate and original summaries.

References

[1] K. Kowsari, D. E. Brown, M. Heidarysafa, K. Jafari Meimandi, , M. S. Gerber, and L. E. Barnes, "Hdltex: Hierarchical deep learning for text classification," in *Machine Learning and Applications (ICMLA), 2017 16th IEEE International Conference on.* IEEE, 2017.

[2] W. McKinney *et al.*, "pandas: a foundational python library for data analysis and statistics," *Python for high performance and scientific computing*, vol. 14, no. 9, pp. 1–9, 2011.

[3] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," 2019.

[4] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019.

[5] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," 2019.

[6] H. T. Le and T. M. Le, "An approach to abstractive text summarization," in *2013 International Conference on Soft Computing and Pattern Recognition (SoCPaR)*, Dec 2013, pp. 371–376.

[7] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking : Bringing order to the web," in *The Web Conference*, 1999. [Online]. Available: https://api.semanticscholar.org/CorpusID:1508503

[8] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," 2019.

[9] C.-Y. Lin, "ROUGE: A package for automatic evaluation of summaries," in *Text Summarization Branches Out*. Barcelona, Spain: Association for Computational Linguistics, Jul. 2004, pp. 74–81. [Online]. Available: https://aclanthology.org/W04-1013