Human-Written vs. Machine-Generated Text Classification

Arti Patel
2329771

Abstract

This project focuses on creating a binary classification that can accurately identify whether a piece of text was written by a human or generated by machines(like chatGPT, cohere, DaVinci, dolly, etc. As AI technology advances we must not overlook the significance of maintaining transparency and trust with readers. It is imperative that we differentiate between human-authored and machine-generated text. Using a dataset of over 100 thousand text samples, this project attempts to create a robust classification model that effectively solves the above problem. The success of this study should enable the user to conclude the authenticity of textual data using the binary classification model generated.

1 Introduction

In today's world, the lines between human-written and machine-generated text are becoming increasingly blurred. With technological advancement, there's a growing trend of utilizing natural language generation (NLG) systems to automate the generation of news articles, reports, and other written content. Computers can now write articles and reports just like humans do. This phenomenon has profound implications across various domains, particularly in journalism and content creation. Maintaining transparency and authenticity in textual content is important for ethical journalism and trust with readers that is, to be honest and clear about who wrote what. However, as AI systems like ChatGPT, Cohere, DaVinci, Dolly, and others become more sophisticated, distinguishing between human-authored and machine-generated text has become a significant challenge. That's why this project is all about building a robust binary classification model that can tell if a piece of writing was done by a person or by a machine.

The dataset for this project consists of JSONL files containing English text samples (63351 human-written text records and 56406 machine-generated records. In total 119757 records are available for training). This dataset will serve as the primary source for training and testing the binary classification model For this project, I will focus on the monolingual track (i.e. the language of the text data is in English). I will experiment with TF-IDF, Doc2vec, and BERT models to come up with a binary classification model with an efficiency as high as possible.

2 Literature review

In the paper [1], the authors propose a single model based on contrastive learning, utilizing about 40% of the baseline model's parameters while achieving comparable performance on the test dataset. The key insight is that a single base model can achieve competitive performance with data augmentation and contrastive learning, eliminating the need for an ensemble of multiple models. In the context of contrastive learning, the model is designed to capture significant differences between machine-generated and human-generated text embeddings. It employs a shared encoder for positive and negative data pairs, using a contrastive learning loss formulation to train embeddings effectively. A classification loss is also utilized for downstream tasks, employing a simple linear layer classifier on top of the embeddings. The model is trained using the AdamW optimizer, with a focus on distinguishing between human and machine-generated text. It achieves optimal results with a maximum sentence length of 256 words, showcasing its adaptability and effectiveness in identifying machine-generated text even in larger documents. Overall, the research highlights the potential of contrastive learning techniques to develop efficient and accurate models for text classification tasks, particularly in distinguishing between human and machine-generated text.

The Term Frequency Inverse Document Frequency (TF-IDF) algorithm was used to achieve excellent results in papers [2] and [3]. The TF-IDF algorithm has two parts: TF part considers the frequency of a word and the IDF part figures out the number of files in which this word can be found; these two parts are used to count the word weight.

Doc2vec was proposed as an extension to Word2vec in the paper [4]. Doc2vec was found to perform well. The authors also recommended hyper-parameters for general-purpose application that they calculated empirically. Similar results were obtained in paper [5]. The authors commented that since the closeness between the sentences and not that between the words was being considered, Doc2vec had a greater advantage over Word2vec.

The research [6] utilizes RoBERTa-base for all experiments, noting its superior performance as a state-of-the-art (SOTA) model in text processing tasks. The authors introduce two key methodologies to enhance the model's efficiency and effectiveness: Weighted Layer Averaging and Parameter Efficient Tuning with AdaLoRa. Weighted Layer Averaging is employed to address the issue of potentially losing syntactic and lexical information by solely using the [CLS] token from the last layer of RoBERTa. Instead, the authors propose averaging all layer hidden states, with each layer assigned a corresponding weight. These weights are trained alongside the classification task, ensuring that valuable linguistic information is captured and utilized effectively. Parameter Efficient Tuning with AdaLoRa addresses the challenge of catastrophic forgetting and the unnecessary update of all model weights during fine-tuning. AdaLoRa, an adaptation of Low-rank Adapters (LoRA), freezes pretrained model weights and introduces trainable rank decomposition matrices into each layer, significantly reducing the number of trainable parameters. This approach allows for efficient tuning while preventing over-parameterization and improving adaptability to changes in information flow. Overall, the research focuses on optimizing RoBERTa-based models for text classification tasks by leveraging techniques such as Weighted Layer Averaging and Parameter Efficient Tuning with AdaLoRa, ultimately aiming for improved performance, parameter efficiency, and adaptability.

3 Dataset Information

The dataset used in this project has been taken from [7]. The data has been extracted from various sites and search engines. This dataset will serve as the primary source for training and testing the Human-Written vs. Machine-Generated Text classification model. It is important to note that this dataset is balanced, ensuring uniform distribution all over classes. The dataset comprises individual sentences, each representing a single string. Below are the data statistics and data format for this project.

- Training Dataset:
  The statistics about the training dataset can be seen in the table 1.

- Validation Dataset:
  The statistics about the validation dataset can be seen in the table 2.

Source	**Model**		**Total**
	bloomz	**human**	
arxiv	500	500	**1000**
peerread	500	500	**1000**
reddit	500	500	**1000**
wikihow	500	500	**1000**
wikipedia	500	500	**1000**
Total	**2500**	**2500**	**5000**

Table 2: Validation Dataset

- Input data format (an object in the JSON format):

```
{
    id: identifier of the
    ↪  example,
    label: label (human text: 0,
    ↪  machine text: 1,),
    text: text generated by a
    ↪  machine or written by a
    ↪  human,
    model: model that generated
    ↪  the data,
    source: source (Wikipedia,
    ↪  Wikihow, Peerread,
    ↪  Reddit, Arxiv) in the
    ↪  English language
}
```

Source	Model					Total
	chatGPT	**cohere**	**davinci**	**dolly**	**human**	
arxiv	3000	3000	2999	3000	15498	**27497**
peerread	2344	2342	2344	2344	2357	**11731**
reddit	3000	3000	3000	3000	15500	**27500**
wikihow	3000	3000	3000	3000	15499	**27499**
wikipedia	2995	2336	3000	2702	14497	**25530**
Total	**14339**	**13678**	**14343**	**14046**	**63351**	**119757**

Table 1: Training Dataset

4 Methodology

4.1 Machine Learning process

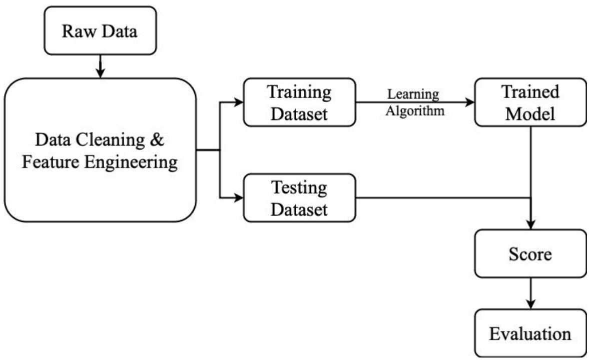

Figure 1: Machine Learning process

1. Data Collection:

   This is the initial phase where relevant data is gathered from various sources like sites, search engines, or data files. The quality and quantity of data collected greatly influence the machine learning model's performance.

2. Data Cleaning and Feature Engineering:

   In this step, the collected data is processed to handle missing values, outliers, and inconsistencies. Feature engineering involves selecting, creating, or transforming features (variables) to improve the model's predictive power. This step is crucial for preparing the data for model training.

3. Training and Testing Dataset:

   The dataset is divided into two parts: the training set and the testing set. The training set is used to train the machine learning model by exposing it to labeled data (input-output pairs). The testing set, which the model hasn't seen during training, is used to evaluate the model's performance and generalization ability.

4. Model Training:

   The machine learning algorithm is applied to the training dataset during this step to learn patterns and relationships between input features and output labels. The model is adjusted iteratively to minimize prediction errors and improve its accuracy.

5. Scoring and Evaluation:

   Once the model is trained, it is evaluated using the testing dataset to assess its performance metrics such as accuracy, precision, recall, F1 score, etc. These metrics help in understanding how well the model generalizes to new, unseen data and whether it meets the desired objectives.

4.2 TF-IDF with Traditional Machine Learning Algorithms

In text classification tasks for machine learning projects, relying solely on word count in a document can lead to certain drawbacks. For instance, common words like stopwords can make the word vectors dense even though they may not carry much significance, while rare words end up sparse and thus may not contribute enough to the model's learning. To address these issues, TF-IDF vectorization is commonly used.

Traditionally, TF-IDF (Term Frequency-Inverse Document Frequency) has been a popular feature representation method for text classification tasks.

The first part is TF, called term frequency. This simply means the number of times the word occur in document divided by the total number of words in the document.

The second part is IDF, which stands for 'inverse document frequency', interpreted like inversed number of documents, in which the term we're interested in occurs.

$tf(t,d)$ is the term frequency is the number of times the term appears in the document.

$$tf(t,d) = \frac{n_{t,d}}{\sum_{k \in d} n_{k,d}}$$

Where t is the term and d is the document. $idf(t,D)$ is the document frequency is the number of documents 'd' that contain term 't'

$$idf(t,D) = log\frac{N}{n_t}$$
$$d \in D, t \in d$$

where N is the total number of documents and n_t is the number of documents containing the term t

The TF-IDF vector,

$$tfidf(t,d,D) = tf(t,d) \times idf(t,D)$$

In this project, I conducted experiments using various machine-learning algorithms for text classification. The algorithms I tested include Naive Bayes, k-Nearest Neighbors, Random Forest, Light GBM, eXtreme Gradient Boosting Random Forest (XGBoost RF), and Logistic Regression [8], [9]. The objective was to compare their performance in terms of accuracy using TF-IDF features and evaluate their effectiveness in classifying text data.

After preprocessing the data and creating TF-IDF features, I trained each algorithm using the respective training sets and then evaluated their performance using confusion matrices to calculate accuracy. The confusion matrices allowed me to analyze how well each algorithm classified instances into their respective classes and identify any misclassifications.

This approach provided insights into the strengths and weaknesses of each algorithm in handling text classification tasks. The results of this experiment are in the results section of this report.

4.3 Doc2Vec Embeddings

Doc2Vec is a natural language processing (NLP) technique that extends the concept of word embeddings to entire documents. It creates dense vector representations for documents, capturing semantic relationships and contextual information. Unlike sparse representations like TF-IDF, Doc2Vec generates continuous, dense vectors that are effective in capturing the meaning of entire documents.

The Doc2Vec model, also known as Paragraph Vector, learns fixed-length feature representations (embeddings) for entire documents. The primary goal is to associate a unique vector with each document in the dataset, enabling the model to understand the semantic similarities and differences between documents. There are two main variations of Doc2Vec: PV-DM (Distributed Memory version of Paragraph Vector) and PV-DBOW (Distributed Bag of Words version of Paragraph Vector)

For this project, I have used the the second one PV-DBOW (Distributed Bag of Words version of Paragraph Vector) [10], [11]. In this approach, the model predicts words independently based solely on the document vector. It treats each document as a "bag of words" and tries to predict words randomly sampled from that bag.

Employing Doc2Vec involves generating document embeddings using PV-DBOW. These embeddings, which are dense vectors representing the semantic meaning of documents, serve as input features to train traditional machine learning algorithms such as Support Vector Machines (SVM), Random Forest, and XGBoost. This approach leverages the semantic understanding captured by Doc2Vec to enhance the performance of the machine learning models in text classification tasks. The results of this experiment are in the results section.

4.4 Experiments with BERT Models

Transfer learning in the context of text classification involves leveraging pre-trained models such as BERT (Bidirectional Encoder Representations from Transformers) and fine-tuning them on specific text data for a particular task [12], [13]. that involves,

The steps involved in using transfer learning with BERT for text classification are as follows:

1. Load the pre-trained BERT model: This involves importing the BERT model architecture and its pre-trained weights. I've used Hugging Face's Transformers BERT implementation for this purpose.

2. Fine-tune the model: Prepared text classification dataset, tokenized the text, and created input sequences suitable for BERT. Then, feed these sequences into the pre-trained BERT model and fine-tune its weights using techniques like gradient descent optimization and backpropagation. The fine-tuning process typically involves multiple epochs of training.

3. Evaluation: After fine-tuning, evaluate the performance of the fine-tuned BERT model on a separate validation or test dataset. Measure metrics such as accuracy, precision, recall, F1 score, etc., to assess how well the model performs on the text classification task.

5 Results

5.1 Data Visualization

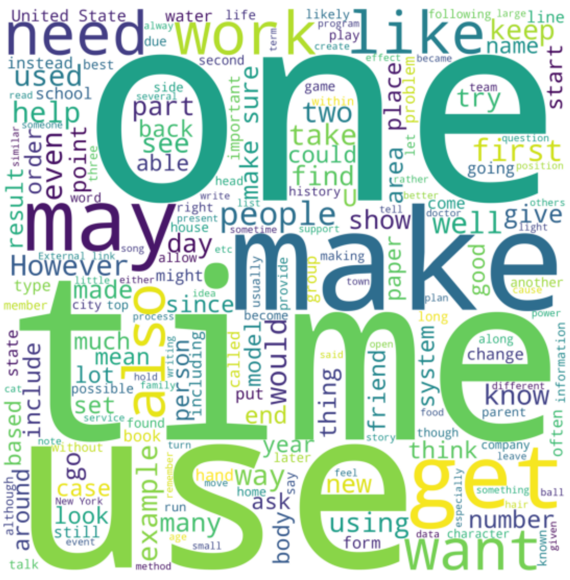

Figure 2: Human-text Word Cloud

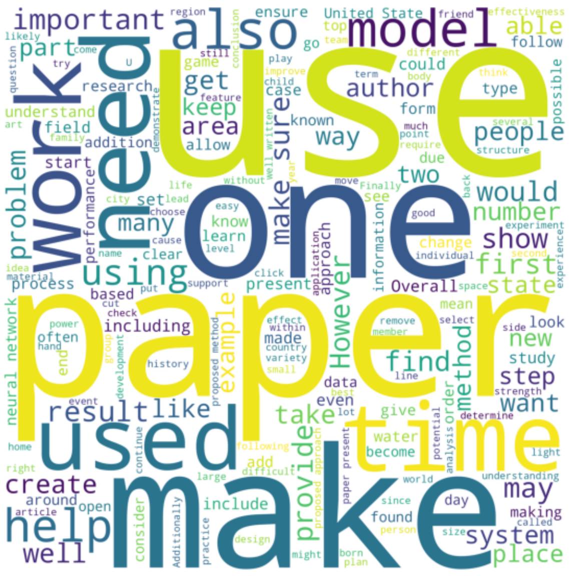

Figure 3: Machine-text Word Cloud

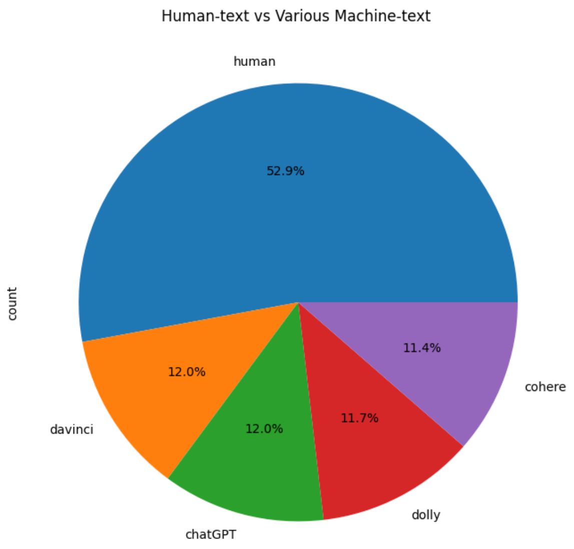

Figure 4: Human-text vs. various Machine-text

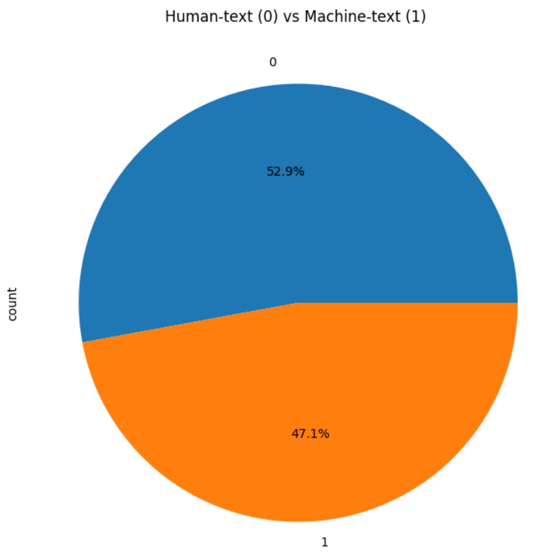

Figure 5: Human-text vs. Machine-text

5.2 TF-IDF with Traditional Machine Learning Algorithms

In this project, I've explored the TF-IDF feature selection technique along with six different machine learning algorithms to classify the data. The algorithms considered were Naive Bayes, Logistic Regression, Light GBM(Light Gradient Boosting Machine), KNN(k-nearest neighbors), eXtra Gradient Boosting(XGboost), and Random Forest. .

Algorithm	Training Accuracy	Validation Accuracy
Naive Bayes	$70.28 \approx 70\%$	$54.50 \approx 55\%$
Logistic Regression	$89.23 \approx 89\%$	$64.64 \approx 65\%$
LightGBM	$78.94 \approx 79\%$	$60.42 \approx 60\%$
KNN	$70.75 \approx 71\%$	$54.22 \approx 54\%$
XGBoost	$91.63 \approx 92\%$	$67.18 \approx 67\%$
Random Forest	$79.63 \approx 80\%$	$60.04 \approx 60\%$

Table 3: TF-IDF

Based on the results obtained during training and evaluation, we can observe that the XGboost algorithm has highest accuracy of 92% with the TF-IDF feature set, making it the best-performing algorithm for this task. But even after achieving this high accuracy it fails to test the completely new data as we can see lower accuracy for validation dataset.

5.3 Doc2Vec Embeddings

For Doc2Vec Embeddings, five machine learning algorithms were explored: Logistic Regression, DT (Decision Tree), Random Forest, KNN (k-nearest neighbors), SVC (Support Vector Classifier). Using the Gensim library, over 10 epochs, 100 dimensions were created for each text data. These data points were then fed into the aforementioned algorithms.

Algorithm	Training Accuracy	Validation Accuracy
Logistic Regression	$73.96 \approx 74\%$	$49.4 \approx 49\%$
Decision Tree	$65.54 \approx 66\%$	$56.44 \approx 56\%$
Random Forest	$75.91 \approx 76\%$	$49.52 \approx 50\%$
KNN	$70.34 \approx 70\%$	$50.48 \approx 50\%$
SVC	$81.59 \approx 82\%$	$49.44 \approx 49\%$

Table 4: Doc2Vec

Overall, these results suggest that Doc2Vec embeddings are providing reasonable accuracy with some algorithms like Random Forest and SVC achieving higher accuracy rates, while others like Decision Tree show lower accuracy. It's important to consider that validation accuracy is also a crucial metric to assess model generalization. but here we have very low accuracy with the Doc2Vec embedding.(possible reason could be validation dataset has completely new data and that is textual data pattern(with a different LLM model) is different than the training dataset)

5.4 Experiments with BERT Models

BERT sequence classification model trained with different maximum sentence lengths and using the AdamW optimizer with a learning rate of **2e-5** and epsilon of **1e-8**, with a batch size of **32**, are as follows:

Max Sentence Length	Training Accuracy	Validation Accuracy
50	86%	$71.30 \approx 71\%$
128	89%	$74.18 \approx 74\%$
256	90%	$75.46 \approx 75\%$
512	92%	$77.44 \approx 77\%$

Table 5: Bert

1. **Effect of Max Length:** As the maximum sequence length increases from 50 to 512, both training and validation accuracies improve. This is expected as longer sequences can capture more context and information, potentially leading to better model performance.

2. **Training Accuracy vs. Validation Accuracy:** There's a noticeable gap between training and validation accuracies, especially for longer sequences. This suggests that the model may be overfitting to the training data, particularly with longer inputs. Regularization techniques or additional data augmentation could help mitigate overfitting.

3. **Batch Size:** Using a batch size of 32 is a common choice and seems appropriate for this task. Adjusting the batch size could influence training dynamics, convergence speed, and memory requirements, but it appears to be well-suited in this case.

4. **Optimizer and Learning Rate:** After experimenting with different optimizers and learning rates, AdamW with a learning rate of 2e-5 and epsilon of 1e-8 is a better configuration for fine-tuning BERT models for this task.

Overall, the results demonstrate the effectiveness of BERT for sequence classification tasks, with higher accuracies achieved on longer sequences. Fine-tuning hyperparameters and addressing overfitting concerns could further enhance the model's performance and generalization ability.

6 Challenges

6.1 Inconsistent Model Performance

One challenge encountered was the inconsistency in model performance between the training and validation datasets, particularly with TF-IDF(Term Frequency-Inverse Document Frequency) and Doc2Vec. During the training phase, these models exhibited promising results, indicating a good understanding of the data and its patterns. However, when tested on the validation dataset, their performance dropped significantly, yielding poor results. This inconsistency in performance suggests that these models might have overfit to the training data, failing to generalize well to unseen data. One possible reason for this could be the difference in the language models used for training and testing, leading to a lack of adaptability and robustness in the models' predictions.

6.2 Computation and Resource Demands

The implementation of the BERT (Bidirectional Encoder Representations from Transformers) model posed substantial computational challenges. BERT, being a deep and complex model, requires a significant amount of computational power to train and evaluate effectively. This high computational demand translates into longer processing times and increased costs, making it a resource-intensive task. The extensive computations also contribute to longer development cycles, hindering rapid iteration and experimentation with different model configurations or hyperparameters. Managing these computational demands effectively while optimizing the model's performance becomes crucial but challenging, as it requires balancing computational resources, time constraints, and budget considerations.

7 Conclusion

This project focused on exploring the efficacy of TF-IDF, Doc2Vec, and BERT models for binary classification tasks distinguishing between human-generated and machine-generated text using a monolingual dataset. While BERT showed promising results, challenges were encountered with TF-IDF and Doc2Vec in terms of performance inconsistency between training and validation datasets.

8 Future work

To address the challenges faced, future work will involve shuffling data between the training and validation datasets to enhance the generalization capabilities of TF-IDF and Doc2Vec models. This approach aims to improve model robustness and performance across different datasets.

Additionally, the project will extend its scope by experimenting with a multilingual dataset. This expansion will enable the exploration of model performance across various languages and enhance the project's applicability in diverse linguistic contexts. Furthermore, instead of binary classification, the project will delve into classifying text into human-generated and specific Language Model (LLM) names, providing more nuanced insights into text classification tasks. These advancements will contribute to a more comprehensive understanding of model performance and applicability in real-world scenarios.

References

[1] S. R. Dipta and S. Shahriar, "Hu at semeval-2024 task 8a: Can contrastive learning learn embeddings to detect machine-generated text?" 2024.

[2] A. A. Hakim, A. Erwin, K. I. Eng, M. Galinium, and W. Muliady, "Automated document classification for news article in bahasa indonesia based on term frequency inverse document frequency (tf-idf) approach," in *2014 6th International Conference on Information Technology and Electrical Engineering (ICITEE)*, 2014, pp. 1–4.

[3] "Classification of movie reviews using term frequency-inverse document frequency and optimized machine learning algorithms — peerj.com," https://peerj.com/articles/cs-914/, [Accessed 03-05-2024].

[4] J. H. Lau and T. Baldwin, "An empirical evaluation of doc2vec with practical insights into document embedding generation," 2016.

[5] I. R. Hendrawan, E. Utami, and A. D. Hartanto, "Comparison of word2vec and doc2vec methods for text classification of product reviews," in *2022 6th International Conference on Information Technology, Information Systems and Electrical Engineering (ICITISEE)*, 2022, pp. 530–534.

[6] A. Datta, A. Chandramania, and R. Mamidi, "Semeval-2024 task 8: Weighted layer averaging

roberta for black-box machine-generated text detection," 2024.

[7] "GitHub - mbzuai-nlp/SemEval2024-task8: SemEval2024-task8: Multidomain, Multimodel and Multilingual Machine-Generated Text Detection — github.com," https://github.com/mbzuai-nlp/SemEval2024-task8/tree/main.

[8] E. Molina, "A Practical Guide to Implementing a Random Forest Classifier in Python — towardsdatascience.com," https://towardsdatascience.com/a-practical-guide-to-implementing-a-random-forest-classifier-in-python-979988d8a263, [Accessed 01-05-2024].

[9] G. Bedi, "Simple guide to Text Classification(NLP) using SVM and Naive Bayes with Python — bedigunjit," https://medium.com/@bedigunjit/simple-guide-to-text-classification-nlp-using-svm-and-naive-bayes-with-python-421db3a72d34, [Accessed 01-05-2024].

[10] D. Mishra, "DOC2VEC gensim tutorial — mishra.thedeepak," https://medium.com/@mishra.thedeepak/doc2vec-simple-implementation-example-df2afbbfbad5, [Accessed 01-05-2024].

[11] N. V. Otten, "Practical Guide To Doc2Vec & How To Tutorial In Python," https://spotintelligence.com/2023/09/06/doc2vec/, [Accessed 01-05-2024].

[12] K. Pham, "Text Classification with BERT — khang.pham.exxact," https://medium.com/@khang.pham.exxact/text-classification-with-bert-7afaacc5e49b, [Accessed 01-05-2024].

[13] J. Alammar, "The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning) — jalammar.github.io," https://jalammar.github.io/illustrated-bert/, [Accessed 01-05-2024].