Success of AI Writer

Ahmet Emre Usta[1] **Huseyin Yigit Ulker**[1]

Abstract

Plagiarism detection is a crucial task in academic and professional settings as it helps to identify and track the use of content from other sources without proper attribution. In this study, we propose a plagiarism detection method using a combination of the state-of-the-art language models RoBERTa, BERT, DeBERTa, and ALBERT. Our method utilizes the transformer-based models RoBERTa, BERT, DeBERTa, and ALBERT, which have been shown to have strong performance on a variety of natural language processing tasks. The models are combined with a bidirectional long short-term memory (Bi-LSTM) neural network, which is able to process input sequences in both forward and backward directions, allowing the model to capture long-range dependencies in the data. The proposed method is trained on the SNLI dataset and evaluated using standard metrics. The results indicate that the combination of these state-of-the-art models with a Bi-LSTM is able to achieve high accuracy in detecting plagiarism in the SNLI dataset. Our models RoBERTa, BERT, DeBERTa, and ALBERT, which have achieved success rates of 90%, 90%, 91%, and 83% in the SNLI dataset respectively. Additionally, the proposed method can handle a variety of text types and languages, making it a promising solution for detecting plagiarism in academic and other settings. The main goal of our research is to detect plagiarism using Artificial Intelligence (AI) techniques. But our main goal is detecting plagiarism which is done by AI.

1. Introduction

Artificial intelligence (AI) writers are gaining popularity because of their efficiency and ability to provide powerful analytics that help predict and optimize content. One popular AI writing tool is Generative Pre-trained Transformer 3 (GPT-3), which can create new sentences and paragraphs based on a few input statements and perform tasks such as rephrasing content, spelling and grammar checking, researching topics, generating ideas, completing forms, and formatting documents. While AI writers can be useful to students, the possibility of plagiarism is a concern. Plagiarism detection programs such as Turnitin and CopyLeaks aim to prevent plagiarism; however, the effectiveness of these programs against AI-generated text is unclear. To address this issue, researchers plan to develop AI models for plagiarism detection using existing datasets and manually paraphrase articles with AI writers to determine how existing models handle duplicate data. It is also suggested that future work could involve retraining and testing existing models using transfer learning with a larger dataset of AI-generated texts. In recent years, there has been a shift towards the use of natural language processing (NLP) techniques for plagiarism detection, as they offer the potential for more accurate and efficient detection. One such NLP approach is the use of a combination of RoBERTa, BERT, DeBERTa, AlBERT pre-trained language models, and a BiLSTM network. BiLSTM networks, on the other hand, are a type of recurrent neural network (RNN) that can process sequential data in both forward and backward directions, allowing for the capture of contextual information from both past and future words.

In this article, we explore the use of AI for plagiarism detection and discuss its capabilities. We also delve into the evaluation and limitations of this approach and discuss potential future developments in the field.

2. Related Work

The need for plagiarism detection programs has increased with technological developments and easier access to data, so many searches have been carried out in this area. Some of these methods can be found in [1] and [7].

In [1] Zakiy Firdaus Alfikri, Ayu Purwarianti created a plagiarism detection model using Support Vector Machines (SVMs) and Naive Bayes methods.In the method, word similarity, fingerprint similarity, hidden semantic analysis similarity and word pair learning features were used. Plagiarism was detected using SVM and Naive Bayes algorithms. As a result, SVM and Naive Bayes achieved a success rate of 92% and 54%, respectively.

In [2] suggested by Chien-Ying, C., Jen-Yuan, Y. , Hao-Ren, K. In this research, the use of synonyms, which is the method that people use most when plagiarism, is discussed in determining the type of plagiarism. For this in

the research, they offer three solutions for synonym recognition that make use of WordNet synonyms. WordNet groups nouns, verbs, adjectives, and adverbs into cognitive synonym clusters called synsets11 [5]. These clusters are then measured for similarity using the Jaccard's coefficient.

In [3], suggested by Mostafa Hambi and Faouzia Benabbou. In this research, 3 deep learning models were used for plagiarism detection: Doc2vec, Siamese Long Short-term Memory (SLSTM) and Convolutional Neural Network (CNN). This study also includes two text comparisons, online comparison and using an intern corpus for a comparison. In the two text comparison, the two documents will be preprocessed and converted to a list of vectors with doc2vec model [7]. The system will detect later if the input documents are similar or not using SLSTM .Using the CNN model, the probability of each type of plagiarism (copy-paste,paraphrasing,false references ,translation,ideas) is calculated.The model in this study can detect not only whether there is plagiarism, but also the probability of plagiarism types.

In [4] Ahmed Hamza Osman, Naomie Salim, Mohammed Salem Binwahlan,Rihab Alteeb and Albaraa Abuobieda introduced the plagiarism detection system using the SRL method. This research, it converts the suspect and original document into arguments based on the position of each term in the sentence using SRL [5] . Next, the proposed method compares the arguments of the doubtful sentences with the similar arguments of the original sentences. According to the similarity result, plagiarism is analyzed.

Recurrent and recursive neural networks were used in the research in article [6]. First, words were vectorized using GloVe. Plagiarism detection was made using recurrent and recursive neural networks in vectorized sentences. As a result, he obtained F1 scores of 81% and 33% in train and test data, respectively.

3. The Approach

3.1. Natural Language Inference

Natural language inference (NLI) is a task in natural language processing (NLP) that allows to determine the semantic relationship between two expressions or pieces of text. In NLI, the goal is often to classify the relationship between the premise and hypothesis as one of these three categories:

Entailment: This indicates that the hypothesis is logically implied by the premise.

Contradiction: This indicates that the premise conflicts with the theory.

Neutral: This indicates that the premise is not logically implied by the premise or contradicted by it.

3.1.1. Stanford Natural Language Inference Dataset

NLI has been considered the Stanford Natural Language Inference dataset (SNLI) is a sizable dataset of natural language phrase pairings that is annotated with labels indicating whether one statement entailment, contradicts, or is neutral with respect to the other. The premises for the SNLI dataset are derived from the picture captions from the Flickr30k corpus. The employees of Mechanical Turk manually produced the hypothesis in following instruction:

Entailment: Create a different caption for the image that is unquestionably accurate.

Contradiction: Add a different caption to the photo that inaccurately describes the image.

Neutral: Write a caption that has nothing to do with the photo.

There are 570k pairs of English sentences in the SNLI dataset. The dataset also had the "-" label. It denotes that the annotators have not reached a consensus, so we eliminate them during the training and evaluation that follow the literature.

df_train_longer_sentences data set in table 1 was produced by extracting lengthy sentences from the df_train dataset. Sentences longer than 60 characters, the median length of the sentences in the df_train data set.This dataset was produced so that the RoBERTa model, which performs better with lengthy sentences, could use it.

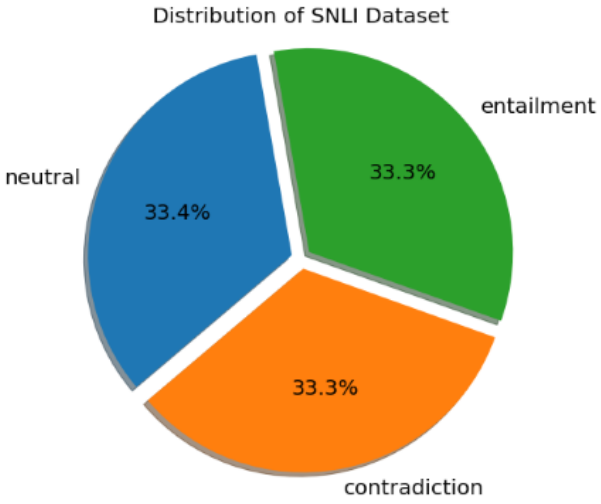

Figure 1. Distribution of SNLI Data Set Labels

Dataset Name	Original Dataset Name	Data Length	Size(MB)	Details
df_train	snli_data	549361	65	SNLI Train Data
df_train_ longer_sentences	snli_data	266582	38	Half of SNLI test data*
df_validation	snli_data	9842	1.2	SNLI Validation Data
df_test	snli_data	9824	1.2	SNLI Test Data
ANLI	anli_data	169265	66.4	All ANLI data (train+validation+test)

Table 1. Dataset Informations

**only longer sentences*

3.1.2. Adversarial Natural Language Inference Dataset

Adversarial Natural Language Inference (ANLI) dataset is a larger-scale dataset based on NLI bench marking compared to other datasets (SNLI, MultiNLI). The main reason for the creation of ANLI dataset is that datasets based on previous NLI comparisons are insufficient for current models(BERT, RoBERTa). For this reason, the technique called Human-And-Model-in-the Loop Entailment Training(HAMLET) is used when creating ANLI data set. This technique includes these steps:

1- A model which called SOTA is created using the NLI based data set.

2- According to the given “premise” and “label” it is aimed to form a ” hypothesis” that the model will label incorrectly.

3- If the model is labeled incorrectly, the premise, label and hypothesis given for validation are checked by two or three people. If the model is confirmed to have made a mistake, the next step is taken.

4- The given sample is added to the data set.

During the creation of ANLI data set, the HAMLET technique was applied 3 times in a row.

Round 1: BERT-LARGE model created with SNLI and MNLI datasets is used.

Round 2: SNLI ,MNLI ,FEWER and the dataset created as a result of the 1st round , RoBERTa model created with datasets is used.

Round 3: SNLI ,MNLI ,FEWER and the datasets created as a result of the 1st and 2nd round , RoBERTa model created with datasets is used.

While creating ANLI data set, as the number of rounds increased, it became more difficult to make the model, which is the second step of the HAMLET model, make incorrect labeling. For example, people averaged 3.4 attempts in round 1 and 6.8 attempts in round 3 [10].

3.2. Model

3.2.1. Transfer Learning

Transfer learning is a machine learning technique that involves using a pre-trained model as a starting point for training a model on a different task. In transfer learning, the objective is to apply the pre-trained model’s knowledge to the new task in order to enhance the model’s performance on the new task.

Feature Extraction

Feature extraction is a technique used in transfer learning that involves using a pre-trained model as a fixed feature extractor, and training a new model using the extracted features as input. The aim of feature extraction is to extract valuable features from the input data that can be utilized to train a model for the new task using the information acquired by the pre-trained model.In transfer learning, selecting a pre-trained model that is well suited to the new activity you wish to do is often the first step. The weights of the previously trained model are then ”frozen” to prevent their updating during training. The pre-trained model is then used to extract the features from the input data, and these features are then passed to the new model. Finally, using the extracted features as input, you train the new model.

Fine Tuning

Fine-tuning is a technique used in transfer learning that involves adapting a pre-trained model to a new task by unfreezing some or all of the layers of the model and training the model on the new task using the pre-trained weights as initialization.In order to increase the model’s performance on the new task, fine-tuning aims to fit the knowledge acquired by the pre-trained model to the unique properties of the new task.

3.2.2. Model Template

Tokenizer

Tokenization is a fundamental process in natural language processing (NLP) that involves dividing a text dataset into discrete units called tokens. These tokens can be words, subwords, symbols, or other smaller units of meaning that can be more easily processed by an NLP model. There are several approaches to tokenization, including word-level tokenization, subword tokenization, and regular expression-based tokenization. The choice of tokenization method can depend on the characteristics of the text and the specific NLP task being performed. The goal of tokenization is to enable more efficient processing of text data by breaking it down into smaller, more manageable units.

Since we are using BERT and BERT-based models in our project, it is important to understand the role of the BERT tokenizer. The BERT tokenizer is responsible for preprocessing the input text so that it can be consumed by the BERT model. This typically involves dividing the input text into individual tokens and encoding these tokens using a vocabulary. Each word in the vocabulary is assigned a unique integer value, or index, and the BERT tokenizer represents each token in the input text as its corresponding index in the vocabulary.

The output of the BERT tokenizer consists of three lists: *input_ids*, *attention_masks*, and *token_type_ids*.

input_ids: List of integers representing the tokenized text, where each integer corresponds to a specific word in the BERT vocabulary.

attention_masks: List of 1's and 0's indicating which tokens in the input text should be considered by the BERT model and which should be ignored.

token_type_ids: List of integers indicating the type of each token in the input text, such as whether it belongs to the first or second sentence of a pair of sentences.

Overall, the BERT tokenizer is an important component of the BERT model because it allows the model to process and understand natural language text. By preprocessing the input text and encoding it in a way that the model can understand, the BERT tokenizer enables the model to learn the relationships between words and their meanings, which is a key component of many NLP tasks.

Layers

The proposed model consists of a relatively small number of layers that are added on top of a pre-trained BERT model. These additional layers are designed to perform specific tasks to complete the desired task of multi-label classification.

The first additional layer is a *Bidirectional* layer, which applies a bidirectional RNN to the input. This layer processes the input in both the forward and backward directions, which can be useful for tasks such as language modeling or machine translation.

Following the Bidirectional layer are two pooling layers: a *GlobalAveragePooling1D* layer and a *GlobalMaxPooling1D* layer. These layers apply average and max pooling, respectively, across the length of the input sequence. This can help reduce the dimensionality of the input and capture the most important features of the input data.

The output of the pooling layers is then concatenated using a *Concatenate* layer. A *Dropout* layer with a dropout rate of *0.3* is then applied to the concatenated output to prevent overfitting.

Finally, the output of the Dropout layer is passed through a *Dense* layer with a *softmax* activation function, which produces the final output of the model.

For the loss function, the model uses categorical cross entropy due to the multi-label classification task. The Adam optimizer is used to update the model parameters during training, and the model's performance is tracked using accuracy.

Layer Type	Output Shape	Parameters
	Input Layers	
input_ids	128	-
attention_masks	128	-
token_type_ids	128	-
	↓	
	TFBert Main Layer	
BERT	-	109482240
	↓	
Bidirectional	128,128	426496
GlobalAveragePooling1D	128	-
GlobalMaxPooling1D	128	-
Concatenate	256	-
Dropout	256	-
Dense	3	771

Table 2. Model Structure Summary

To achieve better performance, the basic structure of the model was kept constant while using different pre-trained models. The pre-trained BERT models are typically trained on input sequences of a fixed length, so only the layers in the middle of the model were modified to accommodate different models. By varying the middle layers, it was possible to experiment with different pre-trained models and evaluate their performance on the task at hand.

In the following sections, we will discuss the significant

differences between the various pre-trained models that were evaluated.

3.2.3. Pretrained Models

We utilized the *"transformers"* library from Hugging Face to obtain pre-trained models for our experiments. The training process was carried out on Google Colab, using the Pro package. While we do not have a specific recollection of the GPU models that were used, it is likely that the NVIDIA A100 40 GB model was utilized for a significant portion of the experiments.

BERT

The BERT model is a pre-trained model developed by Google that has been trained on a large corpus of English text. This specific version, called "bert-base-uncased," uses the uncased version of the BERT vocabulary, meaning that it is not sensitive to the case of the input text. The model also comes with a pre-trained tokenizer that can be used to divide the input text into tokens, which are then encoded as integers using the BERT vocabulary.

In order to train the model, the BERT team used a technique called "masked language modeling," in which a portion of the input tokens are randomly masked and the model is trained to predict the original values of the masked tokens. The model is also trained to perform "next sentence prediction," in which it is given a pair of sentences and must predict whether the second sentence follows the first. These tasks are designed to pre-train the model to understand the relationships between words and their meanings, which can then be fine-tuned for a specific downstream task.

During the training of our model, we first froze the trainable parameters of BERT and used it for feature extraction, which took approximately *36 minutes* for *2 epochs*. We then proceeded to the fine-tuning process, which took *67 minutes* for *2 epochs*. For the feature extraction process, we achieved a training loss of *0.53* and a validation loss of *0.43*, with training and validation accuracies of *0.78* and *0.83*, respectively. In the fine-tuning process, we achieved a training loss of *0.29* and a validation loss of *0.26*, with training and validation accuracies of *0.89* and *0.90*, respectively.

Model Name	bert-base-uncased
Train Datasets	df_train
Val Dataset	df_validation
Max String Length	128
Batch Size	32
Epoch	2
Total Parameters	109,909,507

Table 3. BERT Training Parameters

RoBERTa

roberta-base is a pre-trained language model that was developed using the RoBERTa (Robustly Optimized BERT) training method. This method is based on BERT but utilizes a larger dataset, longer training time, and a more robust training method, which often leads to improved performance on natural language processing tasks. Like BERT, roberta-base includes a pre-trained tokenizer that can be used to tokenize the input text and encode it using the RoBERTa vocabulary.

roberta-base was trained using a variant of the masked language modeling task used to train BERT. In this variant, the input text is divided into blocks, and a random portion of the tokens within each block is masked. The model is then trained to predict the original values of the masked tokens, allowing it to learn contextual representations of the input text that can be useful for various natural language processing tasks.

One key difference between BERT and RoBERTa is the training dataset. RoBERTa was trained on a larger dataset than BERT, which can help the model learn more generalizable representations of the input text. RoBERTa was also trained for a longer period and used a more robust training method, which can further enhance its performance.

During the training of our model, we followed the same procedure as with BERT and first froze the trainable parameters of roberta-base for feature extraction, which took approximately *34 minutes* for *4 epochs*. We then moved on to the fine-tuning process, which took *64 minutes* for *4 epochs*. For this model, we used a different dataset for training. We chose to use longer sentences from our original dataset, as we found in our later experiments that longer sentences were more applicable to real-life situations.

In the feature extraction process, we achieved a training loss of *0.52* and a validation loss of *0.43*, with training and validation accuracies of *0.79* and *0.83*, respectively. During the fine-tuning process, we achieved a training loss of *0.22* and a validation loss of *0.27*, with training and validation accuracies of *0.91* and *0.91*, respectively.

Model Name	roberta-base
Train Datasets	df_train_longer _sentences
Val Dataset	df_validation
Max String Length	128
Batch Size	32
Epoch	4
Total Parameters	125,072,899

Table 4. RoBERTa Training Parameters

AlBERT

albert-base is a pre-trained language model developed using the ALBERT (A Lite BERT) training method. ALBERT is based on BERT but is designed to be more lightweight and efficient while still maintaining strong performance on natural language processing tasks. Like BERT and RoBERTa, albert-base includes a pre-trained tokenizer that can be used to tokenize the input text and encode it using the ALBERT vocabulary.

albert-base was trained using a variant of the masked language modeling task used for training BERT and RoBERTa. In this variant, the input text is divided into blocks and a random portion of the tokens within each block are masked. The model is then trained to predict the original values of the masked tokens, allowing it to learn contextual representations of the input text that can be useful for various natural language processing tasks.

One key difference between ALBERT and BERT is the number of parameters in the model. ALBERT has significantly fewer parameters than BERT, making it more efficient to train and deploy. ALBERT also introduces a new training method called "parameter-reduced training," which further reduces the number of parameters in the model while still maintaining strong performance. This makes ALBERT particularly well-suited for applications where efficiency is a concern.

We decided to use albert-base for our experiment due to its focus on efficiency. We trained the model for both feature extraction and fine-tuning. Unfortunately, we have lost the training process information (loss, accuracy, time, etc.). We mention the saved parameters in the table below. However, we can say that the model we trained with the shortest training time was AlBERT. We will provide test results in a later section.

Model Name	albert-base
Train Datasets	df_train
Val Dataset	df_validation
Max String Length	128
Batch Size	32
Epoch	8
Total Parameters	12,110,851

Table 5. AlBERT Training Parameters

DeBERTa

deberta-v3-xsmall is a pre-trained language model developed using the DeBERTa (Denoising BERT) training method. DeBERTa is based on BERT and is designed to be more robust to noise and corruption in the input text. Like BERT, RoBERTa, and ALBERT, deberta-v3-xsmall includes a pre-trained tokenizer that can be used to tokenize the input text and encode it using the DeBERTa vocabulary.

deberta-v3-xsmall was trained using a variant of the masked language modeling task used for training BERT, RoBERTa, and ALBERT. In this variant, the input text is divided into blocks and a random portion of the tokens within each block are masked. The model is then trained to predict the original values of the masked tokens. However, DeBERTa introduces additional noise and corruption to the input text during training, which helps the model learn more robust representations of the input.

One key difference between DeBERTa and other pre-trained languages models like BERT, RoBERTa, and ALBERT is its ability to handle noise and corruption in the input text. DeBERTa is specifically designed to be more robust to such noise and corruption, making it potentially more effective on tasks where the input text may be noisy or incomplete. DeBERTa also introduces a new training method called "denoising autoencoding," which helps the model learn more robust representations of the input text.

During the training of our model, we followed the same procedure as before and first froze the trainable parameters of deberta-v3-xsmall for feature extraction, which took approximately *4 hours* for *4 epochs*. We then moved on to the fine-tuning process, which took *5 hours and 30 minutes* for *4 epochs*. For this model, we used a larger dataset for training, including the entire *ANLI dataset* in addition to the SNLI data.

In the feature extraction process, we achieved a training loss of *0.50* and a validation loss of *0.36*, with training and validation accuracies of *0.79* and *0.86*, respectively. During the fine-tuning process, we achieved a training loss of *0.28* and a validation loss of *0.23*, with training and validation accuracies of *0.89* and *0.91*, respectively.

Model Name	deberta-v3-xsmall
Train Datasets	df_train + ANLI
Val Dataset	df_validation
Max String Length	128
Batch Size	32
Epoch	4
Total Parameters	70,912,771

Table 6. DeBERTa Training Parameters

3.2.4. Model Results

SNLI Test Results

The table below represents the results of evaluating the performance of four pre-trained language models on an SNLI test dataset.

The test time for each model is given in seconds and repre-

Model Name	df_test_time (second)	df_test_loss(x10^{-4})	df_test_acc(x10^{-4})	Test Environment
bert-base-uncased	22	2778	9022	Colab Pro
roberta-base	17.1	2749	9053	Colab Pro
albert-base	17.7	4179	8391	Colab Pro
deberta-v3-xsmall	23	2535	9108	Colab Pro

Table 7. SNLI Data Test Result

Model Name	ANLI_test_time(second)	ANLI_test_loss(x10^{-4})	ANLI_test_acc(x10^{-4})	Test Environment
bert-base-uncased	364	11211	6025	Colab Pro
roberta-base	277	15230	5424	Colab Pro
albert-base	285	11476	4985	Colab Pro
deberta-v3-xsmall	232	2199	9226	Colab Pro

Table 8. ANLI Data Test Result

sents the time it took to run the model on the test dataset. The test loss and test accuracy are given in $x10^{-4}$ and represent the model's performance on the test dataset. A *lower test loss* and *higher test accuracy* indicate better performance.

Overall, it appears that bert-base-uncased and deberta-v3-xsmall had the *best performance*, with test losses of *2778* and *2535* $x10^{-4}$, respectively, and test accuracies of *9022* and *9108* $x10^{-4}$, respectively. roberta-base and albert-base also had a relatively good performance, with test losses of *2749* and *4179* $x10^{-4}$, respectively, and test accuracies of *9053* and *8391* $x10^{-4}$, respectively. All four models were tested in the Google Colab Pro environment.

ANLI Test Results

The above table presents the results of evaluating the performance of four pre-trained language models on the ANLI test dataset. The time it took for each model to run on the ANLI test dataset is shown in seconds. The ANLI test loss and ANLI test accuracy, which represent the model's performance on the ANLI test dataset, are given in $x10^{-4}$. A *lower ANLI test loss* and *higher ANLI test accuracy* indicate better performance.

Overall, it appears that deberta-v3-xsmall had the best performance, with an ANLI test loss of *2199* $x10^{-4}$ and an ANLI test accuracy of *9226* $x10^{-4}$. However, this result should be interpreted with caution as the DeBERTa model was trained on the ANLI dataset, which could have artificially inflated its performance. It is also worth noting that deberta-v3-xsmall had the shortest ANLI test time, at *232 seconds*. This may be more relevant to the goals of the experiment, as the researchers were primarily interested in the running times of the models. bert-base-uncased and roberta-base also had a relatively good performance, with ANLI test losses of *11211* and *15230* $x10^{-4}$, respectively, and ANLI test accuracies of *6025* and *5424* $x10^{-4}$, respectively. albert-base had the lowest ANLI test accuracy, at *4985* $x10^{-4}$. All four models were tested in the Google Colab Pro environment.

Overall, these results suggest that the deberta-v3-xsmall model had the best performance in terms of ANLI test loss and ANLI test accuracy, while also having the shortest ANLI test time. However, it is important to consider the potential impact of training the DeBERTa model on the ANLI dataset when interpreting these results.

3.3. Similarity Checker

Because our study focuses on semantic similarity, we compare sentences rather than words while comparing the two texts. We used the similarities between sentences to determine the overall similarity of the texts.

We classified the model outputs of the phrases with the "entailment" label as comparable while computing the overall similarity ratio and identified the "entailment" ratio as the similarity ratio. The probability of being a "entailment" is averaged out to determine the text's overall similarity ratio.

The computation was made by multiplying the total number of sentences because we compared the sentences in the two texts to obtain the overall similarity of the text from the sentences. This causes the model to execute more slowly.

3.3.1. Soft Cosine Similarity

Soft cosine similarity is a metric for comparing the similarity of two texts that takes into consideration both the semantic similarity of the words used and the contextual similarity of those words. It resembles the conventional cosine similarity measure in that it computes the cosine of the angle between two document vectors in a multidimensional space, where each dimension stands for a phrase in the text. The soft cosine similarity metric uses a matrix of term similarity to

take into account the possibility that words in the documents may have similar meanings but be represented by different terms.

Using the 'fasttext' package, a sizable collection of text data from news stories and Wikipedia was used to train the FastText-Wiki-News-Subwords-300 model. The model handles out-of-vocabulary (OOV) words more successfully than traditional word-based models because it depicts words as a combination of subwords or character n-grams.

In order to simplify time complexity in our work, we adopted the soft-cosine analogy. This is due to the fact that the neural network model takes longer than the soft-cosine similarity to determine the similarity of two texts. We first determined the soft-cosine similarity values between the two texts in order to shorten the model's execution time. We determined the similarity ratio with the neural network model if it is more than a predetermined threshold. We chose 0.764690 as the limit value since it is the size of the 0.35 percentile of cosine similarity values in the data set we produced using GPT-3.

4. Experiment

4.1. GPT-3 Data Generating

In academic research, plagiarism detection software is frequently used to assess the originality of the work. We used Assoc. Prof. Mehmet Erkut Erdem's open access articles to paraphrase the articles from GPT-3, Open AI Products, to evaluate the performance of our model. For the article's title, abstract, and introduction parts, we applied the paraphrasing technique. For these operations, we used the GPT-3 model's following prompts:
Title : " Paraphrase the given title, using as few words from the original title as possible while keeping the key points: "
Abstract and Introduction: " Paraphase the following paragraph while keeping scientific details and using as few words from original paragraph. Output must be the longer sizes as input: "

We have uploaded a dataset to Kaggle that consists of 69 articles with their original titles, abstracts, introductions, and URLs, as well as their paraphrased versions. The dataset includes both the original and paraphrased forms of the articles.[11].

4.2. Comparing Models

We used the data set produced by GPT-3 to evaluate the models' performance. The ground truth value of 100% was determined as the similarity ratio between the original text and the text produced by GPT-3. Testing the data generated by each model produced the results given in Table 10. This table presents the results of evaluating four natural language processing models (BERT, RoBERTa, AlBERTa, and DeBERTa) on a specific task. The task involves measuring the similarity between the title, abstract, and introduction of some text. The performance of each model is represented as a percentage.

Parameters	**Title**	**Abstract and Introduction**
model	text-davinci-003	text-davinci-003
temperature	0.85	0.8
max_tokens	dynamic*	dynamic*
top_p	0.7	0.75
frequency_penalty	0	0
presence_penalty	0.4	03
best_of	4	3

Table 9. Parameters Utilized to Generate Data Using GPT-3

**dynamically calculated using input text lengths*

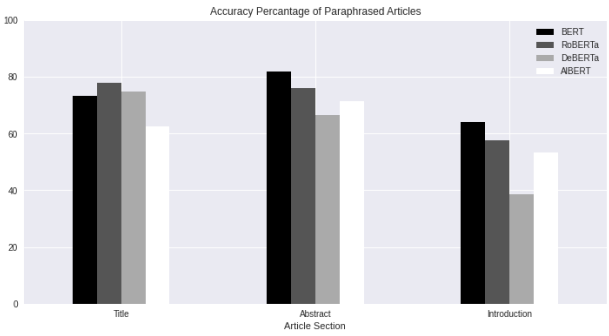

Figure 2. Accuracy Percentage of Models on Paraphrased Articles

According to the table 10 and figure 2, the BERT and DeBERTa models performed relatively well in terms of calculating the similarity of the title, with percentages of 73.3 and 74.6, respectively. The RoBERTa model had the highest percentage (77.8) in the abstract similarity measure. For the introduction similarity measure, BERT had the highest percentage (64.1), while DeBERTa had the lowest (38.5). Based on the models' overall accuracy performance, they can be ranked, considering the different datasets we trained, such as BERT, RoBERTa, AlBERT, and DeBERTa.

According to the results of our evaluation, which considered various datasets on which the models were trained, the overall accuracy of the models can be ranked as BERT, RoBERTa, AlBERT, and DeBERTa. It is important to note that the relative performance of the models may vary depending on the specific task and data they are applied to and that other factors, such as training data size and model architecture, can also impact their performance. However,

	BERT	RoBERTa	AlBERTa	DeBERTa
Title Similarity Percentage	73,3	**77,8**	62,6	74,6
Abstract Similarity Percentage	**81,8**	75,9	71,5	66,4
Introduction Similarity Percentage	**64,1**	57,7	53,3	38,5

Table 10. Model Results for Data Generated with GPT-3

based on the results obtained in our study, the aforementioned ranking represents the general trend in the models' accuracy.

5. Deploying Model

Gradio is an effective tool for deploying machine learning models, especially for natural language processing (NLP) problems. It enables users to quickly develop interactive interfaces for their models that are accessible online via an API. Simply defining the model and giving it to the Gradio API is all that users need to do to deploy a machine-learning model using Gradio. Following that, the API will automatically create an interactive interface that enables users to enter data, evaluate predictions made by the model, and investigate the model's behavior. This can be especially helpful when showcasing the model's capabilities to others and when seeking feedback and ideas.

The model can be accessible by other apps via the internet via an API once it has been Gradio-deployed. Users can easily employ the model's capabilities for a variety of NLP applications thanks to its ease of incorporation into their projects.

Generally speaking, Gradio is an effective instrument for deploying machine learning models, especially for NLP workloads. It enables users to quickly develop interactive interfaces for their models and share them with others in a simple manner.

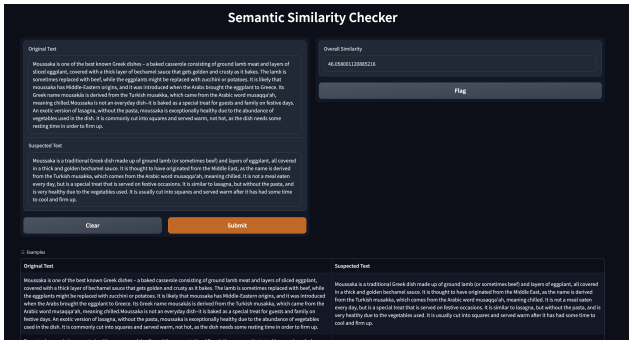

Figure 3. Gradio Demo Application

As shown in Figure 3, we locally deployed our machine learning model using the Gradio platform, which offers several benefits including the ability to create interactive interfaces and API-based access. In the future, we are planning to publish our model on the Hugging Face platform to make it more widely available to other researchers and practitioners in the field of natural language processing (NLP). By sharing our model in this way, we hope to contribute to the ongoing development of NLP techniques and enable others to build upon our work.

6. Conclusion

In this AIN311 course project, we aimed to develop a semantic similarity checker by evaluating the performance of four pretrained language models: BERT, RoBERTa, AlBERTa, and DeBERTa. We first measured the performance of each model on the df_test dataset and found that RoBERTa had the highest df_test accuracy at 9053 10^{-4}, followed by deberta-v3-xsmall at 9108 10^{-4}. We then evaluated the models on the ANLI test dataset and found that while deberta-v3-xsmall had the highest ANLI test accuracy, at 9226 10^{-4}, this result was likely because the ANLI dataset was used as the training set for deberta-v3-xsmall. When considering both the df_test and ANLI test results, RoBERTa and bert-base-uncased had the highest overall performance, with df_test accuracies of 9053 and 9022 10^{-4}, respectively, and ANLI test accuracies of 5424 and 6025 10^{-4}, respectively.

In addition to evaluating the performance of the models on the test datasets, we measured the semantic similarity of the original and paraphrased articles in our dataset. We found that RoBERTa had the highest overall similarity percentage of 77.8% for the paraphrased title. For paraphrased abstracts, BERT had the highest score (81.8%). Again, BERT's most successful model for finding similarities at the paraphrased introduction was 64.1%.

However, one of the major limitations of our study was the relatively small dataset used for training and testing the models. While the df_test dataset consisted of approximately 75,000 examples and the ANLI dataset consisted of approximately 170,000 examples, these datasets may not have been

sufficient to fully capture the complexity and diversity of real-world text. As a result, the models may have struggled to generalize to new examples and achieve high levels of performance in the semantic similarity task.

Overall, our study provides a promising starting point for the development of a semantic similarity checker, but there is still room for improvement. By addressing the limitations of our study and exploring alternative approaches, we believe it may be possible to develop more robust and accurate semantic similarity checkers that can be applied to a wide range of real-world tasks. Our approach is not very common, but we believe that it has potential. With better data, using a proper development environment for better hyperparameter tuning can yield more accurate results. For future work, we will consider these factors.

References

[1] ALFIKRI, Z. F., AND PURWARIANTI, A. "detailed analysis of extrinsic plagiarism detection system using machine learning approach (naive bayes and svm), 2014.

[2] CHIEN-YING, JEN-YUAN, C., AND Y.AND HAO-REN, K. Plagiarism detection using rouge and wordnet. journal of computing, 2(3), 34-44, 2010.

[3] HAMBI, E. M., AND BENABBOU, F. A new online plagiarism detection system based on deep learning in international journal of advanced computer science and applications, 2020.

[4] OSMAN, A. H., SALIM, N., BINWAHLAN, M. S., ALTEEB, R., AND ABUOBIEDA, A. An improved plagiarism detection scheme based on semantic role labeling, 2012.

[5] PAUL, M., AND JAMAL, S. An improved srl based plagiarism detection technique using sentence ranking, 2014.

[6] SANBORN, A., AND SKRYZALIN, J. Deep learning for semantic similarity., 2017.

[7] SANDILYA, N., SHARMA, R., AND MELEET, M. Plagiarism detection using natural language processing and support vector machine, 2022.

[8] RoBERTa and BiLSTM Model, "https://huggingface.co/keras-io/bert-semantic-similarity/blob/main/model.png", 23-01-09

[9] Transfer Learning, "https://ratsgo.github.io/nlpbook/docs/introduction/transfer/",23-01-09

[10] ANLI Dataset, "https://towardsdatascience.com/natural-language- inference-an-overview-57c0eecf6517" , 23-01-09

[11] Paraphrased Articles Using GPT3, https://www.kaggle.com/datasets/aemreusta/paraphrased-articles-using-gpt3 , 2023-01-09.

A. Appendix

Kaggle Dataset Link

https://www.kaggle.com/datasets/aemreusta/paraphrased-articles-using-gpt3

Prpject GitHub Repository

https://github.com/a-emreusta/success-of-ai-writers