Abstract

Detecting the AI-generated percentages of text in assignments and its originality that it should not match with other student work is very important to maintain academic integrity. To do this we calculate similarity score between all the submissions. There are various tools present like ChatGPT, Gemini, Copilot, etc. that students use to complete their assignments so it is very important to distinguish between AI generated and Human written text. In this study, we have used DAIGT dataset that contains AI generated text from various AI models and human written texts with labels and for this TF-IDF and BERT model is used to generate text embeddings for performing the Machine Learning task trained various Machine Learning Classifier such as MLP, LightGBM, CatBoost, Logistic Regression, SVC, Random Forest and KNN with these embedding vectors. For similarity detection we are using TF-IDF, BoW, SBERT, Cosine Similarity and Jaccard Similarity with n- gram for detecting the similarity between the two document. Every model performed well on predicting the dataset testing part but according to our observation MLP has achieved best generalizability and worked well on real time data. For similarity detection part we used Jaccard similarity with n-gram for detecting exact copy paste work is best. At last, we have used flask library to host the web servers for the comprehensive report through the analysis of a ZIP file containing all the assignments.

Table of Content

Chapter No.	Topic	Page No.
	Acknowledgement	iii
	Abstract	iv
1	**Introduction**	1
2	**Literature Survey**	2
3	**Proposed Methodology**	3
	3.1 Vectorization Techniques	3
	3.1.1 TF-IDF Vectorization	4
	3.1.2 BERT & SBERT Embeddings	4
	3.1.3 Bag of Words (BoW)	7
	3.2 Classification models	7
	3.2.1 Multi-layer perceptron (MLP)	7
	3.2.2 LightGBM	7
	3.2.3 CatBoost	8
	3.2.4 Logistic Regression	8
	3.2.5 Support Vector Machine	9
	3.2.6 Random Forest	10
	3.2.7 K-nearest neighbours	10
	3.3 AI-generated text detection	10
	3.4 Similarity detection between the documents	16
	3.5 FLASK application	17
	3.5.1 Workflow Diagram	20

4.	**Experimental Setup**	21
	4.1 Dataset	21
	4.2 Environment	21
	4.3 Technology Stack	21
5.	**Results and Analysis**	22
	5.1 AI Detection Models	22
	5.1.1 BERT Embeddings	22
	5.1.2 TF-IDF Vectorization	24
	5.2 Similarity Detection Performance	24
	5.2.1 Cosine Similarity	24
	5.2.2 Jaccard Similarity	25
	5.2.3 TF-IDF with Cosine Similarity	25
	5.2.4 SBERT	26
	5.2.5 BoW	26
	5.3 FLASK Application Output	26
6.	**Conclusion and Summary**	28
7.	**References**	29

List of Figures

Figure no.	Name	Page no.
3.1	Multi-Layer Perceptron	8
3.2	CatBoost Model	9
3.3	BERT Model for generating embeddings	11
3.4	BERT Model Architecture	15
3.5	Cosine similarity of Vectors	17
3.6	Workflow Diagram	20
5.1	Flask Application Input	27
5.2	Flask Interface Report	27

List of Tables

Table no.	Name	Page no.
5.1	Output using BERT embeddings	23
5.2	Output using TF-IDF vectorization	25

List of Abbreviations

Abbreviations	Full Form
ML	Machine Learning
NLP	Natural Language Processing
MLP	Multi-Layer Perceptron
SVM	Support Vector Machine
SVC	Support Vector Classifier
KNN	K-Nearest Neighbor
TF-IDF	Term Frequency-Inverse Document Frequency
BoW	Bag of Words
BERT	Bidirectional Encoder Representations from Transformers
SBERT	Sentence Bidirectional Encoder Representations from Transformers
AI	Artificial Intelligence

1. Introduction

This comes to our mind that many students uses AI tools like ChatGPT, Copilot, Gemini, etc. for completing their assignments given by their teacher and some students just copy their friend work by direct copy pasting. So, we are going to implement the project “Academic Submission Authenticity Detection” where we address both the issues and help educator to distinguish between the well performing students in academics and can assign marks to each students on the basis of their work. Academic Submission Authenticity Detection has two main sub part which are AI generated text detection and the similarity between two assignment. To make it more user friendly a comprehensive report is shown after submitting the downloaded zipped folder from Google Drive by educator and on educator screen the similarity score of two assignments and AI-generated percentage will get calculated. In this report we are going to discuss the methodology we have used for detecting the AI generated assignment and the similarity between two documents by calculating their similarity score using various vectorization techniques. To take a brief overview of NLP domain and various Machine learning classifiers we will be using many pretrain models used in classification task for AI-generated text detection such as multiple layer perceptron (MLP), LightGBM, CatBoost, logistic regression, support vector machine (SVM), random forest, and K-nearest neighbours (KNN) and these model will be trained from the vectors that will be generated from BERT embeddings and TF-IDF. We are using Term Frequency-Inverse Document Frequency (TF-IDF), Bag of Words (BoW), Sentence Bert (SBERT) for vectorisation of text and n-gram with Jaccard similarity in the calculation of similarity score. For flask server, in upload files educator or user can upload the assignment zipped. Our web-based application extract those files save the folder with different different name for avoiding any collision occurance of same files which is helpful for creating of database of those files too. Our goal is not only to perform on the dataset but main goal will be focusing on real time data too as it is important for an educator to see the real time predictions on the dataset.

2. Literature Survey

We have gone through many latest and some important old research paper for clarifying our concept of BERT embeddings and transformer-based architecture and the concept of self-attention layer. For plagiarism detection in the documents we have seen many methods like cosine similarity Jaccard similarity and also gained information about using them with various vectorization technique like BoW and some are using pretrained LLM model Sentence BERT in the calculation of vectors from the text.

We have also seen the study of using SVM classifier trained on the similarity of sentences for plagiarism detection from one of paper **[1], [3]**. For that, researchers used PAN 2012 and 2013 corpora which consists of a lot of documents created for the evaluation of plagiarism detection. In that particular research 34 sentence similarity features were computed from dataset in feature extraction and they ranked them using feature selection approach (Chi square). After that the most discriminative features selected to train SVM to detect plagiarism or not.

For AI generated text we have seen many researches **[2], [5]** which were using the pretrained machine learning models like SVM, Random Forest with transformer-based BERT model. Also their model combines various layers such as Embedding Layer, Bidirectional LSTM, Transformer Block Conv1D layer. For minimising the loss Adam Optimiser and binary cross entropy loss function are used for the compilation of the model. They have used model checkpoints for saving best performing model and Early Stopping to stop training when validation metrics stop improving.

By going through all the researchs we have decided to use such techniques for our model too. But for differetiating the project we will be applying various models and different vectorization technique. For making it real time project, our goal is not just to perform good on the dataset but to perform on real time data too. So we are also going to handle the task for large documents as its length is limited to 512 token only. So, we will be handling it by making chunks for full document.

3. Proposed Methodology

Let's discuss the methodology we are following to achieve our goal of detecting AI generated assignments and the similarity between two assignments. To discuss these steps we need to convert the text into vector as Machine learning doesn't understands texts it only understands number for the prediction. The vectorisation techniques we have used in AI text detection is Term Frequency-Inverse Document Frequency (TF-IDF) and BERT embedding on our dataset DAIGT.

DAIGT is one of the best dataset we have found on the internet as it contains the AI text from various AI LLM models to make it capable for detecting AI-generated text from different AI models instead of a particular model as students can copy their text from any AI model where our model can lag or gives wrong output in detecting it. This dataset contain a total of 27371 labels of Human written and 17497 labels of AI generated text. The LLM models it contains are *ChatGPT Moth, LLaMA2 Chat, Mistral7B Instruct v2, Mistral7B Instruct v1, Original Moth, Train Essays, LLaMA 70B v1, Falcon 180B v1, Darragh Claude v7, Darragh Claude v6, Radek 500, NousResearch/LLaMA-2-7B-chat-hf, Mistralai/Mistral-7B-Instruct-v0.1, Cohere Command, PaLM Text-Bison1, Radek GPT-4* as per given by providers of the dataset. Out of 5 provided columns *text, labels, prompt name, source, RDizzl3_seven,* we will be using only text and labels for our dataset for prediction purpose and drop other columns as our main tasks is to find percentage of AI generated text not to find the LLM model it comes from.

3.1 Vectorization Techniques:

We have various vectorization techniques in Natural language processing for converting the text into the vectors. Some of the techniques which we are going to use are TF-IDF (Term Frequency-Inverse Document Frequency), BERT(Bidirectional Encoder Representations from Transformers), SBERT (Sentence Bert) and BoW(Bag of Words) and the are defined as:

3.1.1 TF-IDF Vectorization:

TF-IDF (Term Frequency-Inverse Document Frequency) is a popular technique used for converting the text into numerical form (vectors) for machine learning. It helps in finding out the importance of words in a document with respect to a collection of documents (corpus).

- **TF** (Term Frequency): It shows the importance of a word or term in a document and calculates how many times the word has come in that particular document.

TF= Number of occurrence of term in document d / Total number of terms in document d

- **IDF** (Inverse Document Frequency): It tell us that a particular word is how much rare in the whole corpus/documents means the importance of word in the corpus. The rare words will have high importance on the other hands the common words such as "the", "is" has less importances only because of their frequency in the corpus as the are the most frequent words. But also if that rare word is used by other document too then it will be generating high similarity score.wh

IDF=log(Total number of documents /1 + No. of documents containing term)

- **TF-IDF**: It is the multiplication of *TF* and *IDF* of a particular word. If a word coming in a document oftenly it will have high *TF* and also it will have high IDF score means in corpus it is rare which results high TF-IDF score which is differentiating that particular document from others.

TF-IDF= TF *IDF

3.1.2 BERT & SBERT Embeddings:

BERT known as *Bidirectional Encoder Representations from Transformers* is a transformer-based language model developed by Google in 2018. It is used for

creating the vectors from the text but it has a speciality of understanding the context from both left and right simultaneously. BERT process text as input tokens which includes *[CLS], [SEP], [MASK]* and other tokens of sentences. [CLS] token is added at the beginning of every sentence and its output vector is used for classification task as it is basically the summary of the whole sentence. [SEP] token is used for separating the two sentences. [MASK] plays a important role in Masked Language Modelling (MLM) task. It is used to hide the word in input text during training and the task of BERT is to identify that masked word based on its context which helps model to remember the context of sentence.

Talking about how it is made from transformer we have to study what transformer includes. Transformer architecture is deep learning architecture introduced in the paper "Attention is All You Need" by Vaswani et al. in 2017. It was designed to perform the tasks like *machine translation, text generation, and speech recognition* without relying on recurrent or convolutional networks. There are two parts in transformer encoder and decoder. The encoder in the Transformer model is responsible for processing the input sequence and creating a sequence of *contextual embeddings* (word representations) and the decoder part generates the output sequence which includes receiving a output from the encoder along with the previous tokens generated during training, typically used in tasks like machine translation.

BERT model uses the encoder part of the Transformer model which contains bidirectional Self Attention layer (both left-to-right and right-to-left), allowing it to understand the full context, Feed forward neural network which is a type of neural network where information moves in forward direction only i.e. through hidden layer to output layer. Self-Attention is the main part of the encoder where each token in the sequence is represented int the form of three vectors *Key (K), Query (Q), and Value (V).*
Here, Querry represents the word which is trying for gathering the information from

the other words in the sentence for understanding the context. Key represents the information which each word offers to other word. At last, Value is the actual information holding that will be pass when a word attend another word. Self-Attention layer working is quite complex it first find the Dot product of *Querry (Q) and Key (K)* of every word. Then scaling is applied to ensure stable gradient and prevents high large values in the dot product. The result is scale by dividing by the square root of the dimension of the key vector ($\sqrt{d_k}$). After that Softmax function is applied on the scaled dot products to normalize the score into probability distribution for maintaining the attention weight 1. After softmax, the attention weight (from softmax) are used to take a weighted sum of the *Values(V)* for each word. The resulting weighted sum represents how much information each word should incorporate from the other words.

Example: ***"The dog barks loudly."***

- Query for "dog" will look at "the", "dog", "barks", and "loudly" to determine how relevant each of these words is to understanding the meaning of "dog".
- The Key for "barks" will indicate how important "barks" is when "dog" queries it.
- The Value for "barks" will hold the actual embedding of the word "barks," which will be passed to "dog" if it is considered relevant.

Talking about the Sentence-BERT (SBERT) it is a modified version of BERT designed to generate the sentence-level embeddings for task like semantic textual similarity, clustering, and retrieval. Unlike BERT, which uses token-based embeddings, SBERT fine-tunes BERT to generate fixed-size embeddings for entire sentences which makes it more efficient for task requiring sentence level understanding. SBERT uses a siamese network architecture, where two identical BERT models process sentence pairs. This allows it for capturing the semantic meaning between the sentences. It generates a fixed size vector er sentence in the output which contains the overall meaning of sentence that can be used for various tasks like sentence similarity.

3.1.3 Bag of Words(BoW):

It is a simple model or method commonly used in Natural Language Processing to convert textual data in numerical data. Its basic work is frequency of particular word in a document and ignores the word order. Firstly, the text will be divided into unique token and we write their frequency of occurance in tat particular document.

3.2 Classification models:

For classification task we have to find the probability generated from all of the models. The models we are going to use is multi-layer perceptron (MLP), LightGBM, CatBoost, logistic regression, support vector machine (SVM), random forest, and K-nearest neighbours (KNN). Let's see a brief overview of each models:

3.2.1 Multi-layer perceptron (MLP):

Multi-Layer Perceptron is a deep learning artificial neural network model which contains multiple neron layers and every neuron is connected to its next layer. In MLP there are three type of layers input Layer, hidden layer and output layer but the main game is with hidden layer as if we want to increase the model accracy and making our model more complex it will result high accuracy score. It uses activation function such as Sigmoid, ReLU and Tanh for introducing the non-linearity to the model. The Fig 3.1 show exactly how a MLP model works with one hidden layer.

3.2.2 LightGBM: LightGBM, which stands for Light Gradient Boosting Machine, is an open-source machine learning framework made by Microsoft. It uses a special way of learning from decision trees based on histograms to make predictions, mostly for ranking and classification tasks. The LightGBM algorithm gets its unique features from two methods: GOSS (Gradient-based One-Side Sampling) and EFB (Exclusive Feature Bundling). LightGBM is famous for being fast and efficient, plus it has all the good things that XGBoost has.

3.2.3 CatBoost: CatBoost is another open-source algorithm that uses the gradient-boosted decision tree method. It's great for handling big, complex datasets, especially those with categorical features. Unlike other algorithms that need extra steps like one-hot encoding, CatBoost can work directly with these types of variables.

3.2.4 Logistic Regression: Logistic Regression is a tool used to predict the chance of something happening. It looks at how different pieces of information, like one main factor and one or more other factors, relate to each other. This method uses a special S-shaped curve called the logistic function. To figure out the numbers it uses, called coefficients, it often uses a method called maximum likelihood estimation. The result is a score between 0 and 1, which can then be turned into a yes-or-no answer.

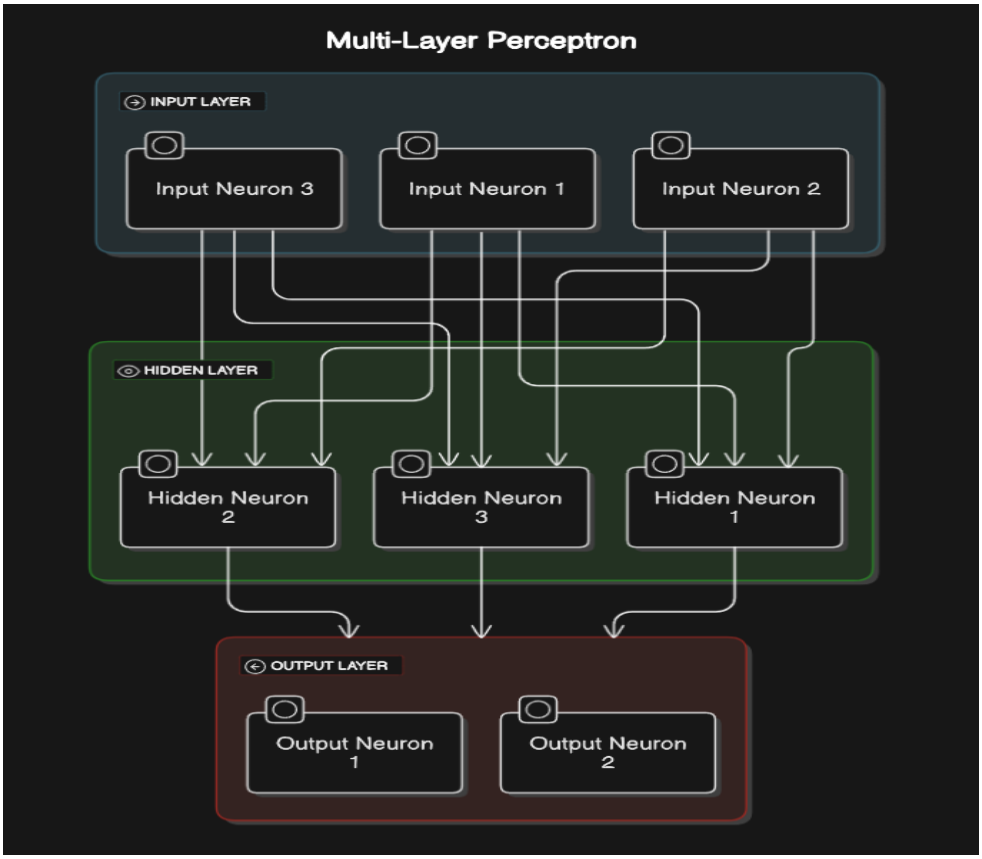

Fig 3.1 Multi-Layer Perceptron

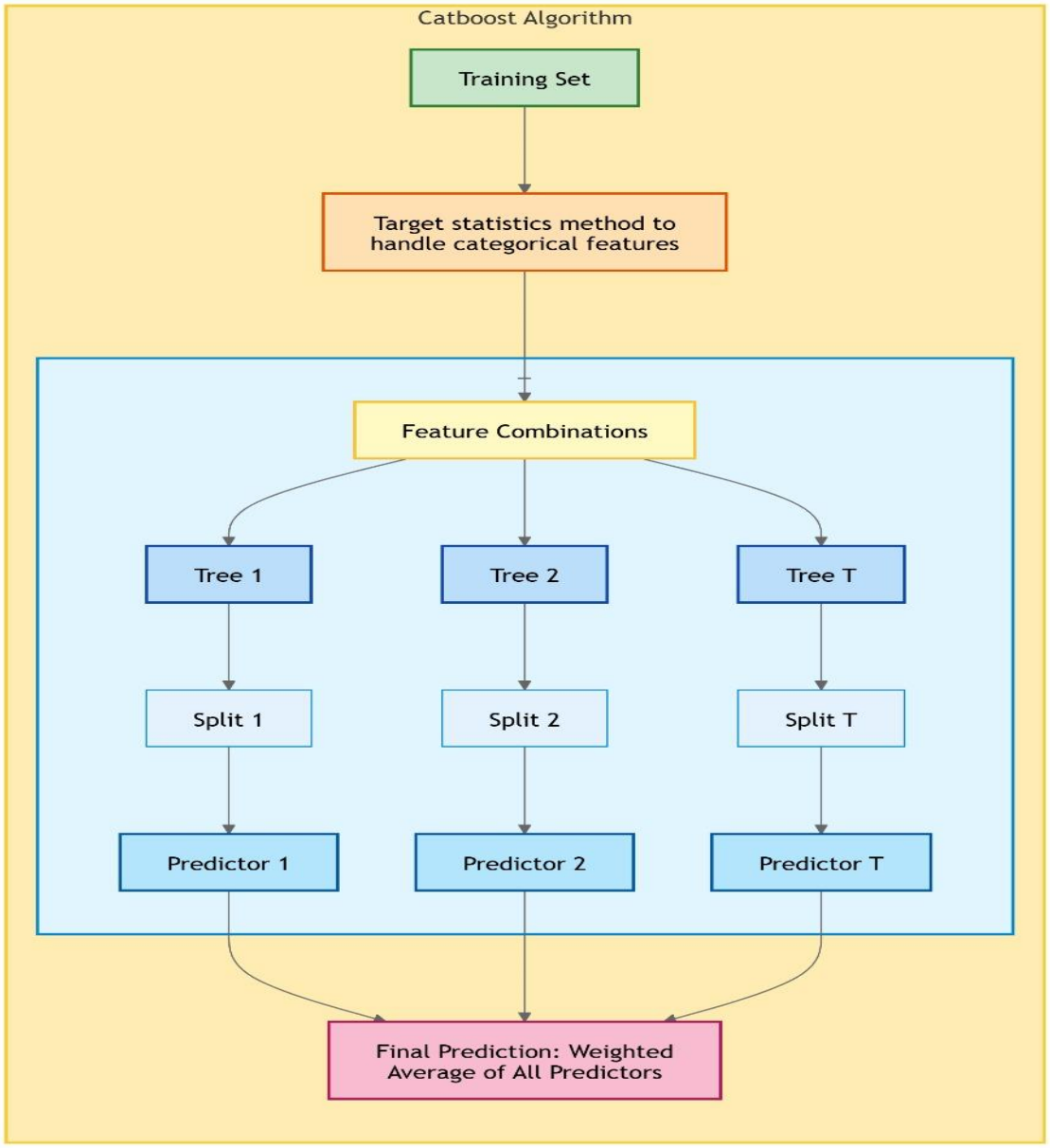

Fig 3.2 CatBoost Model

3.2.5 Support Vector Machine: A Support Vector Machine (SVM) is a supervised learning algorithm that aims to solve both classifications and regression tasks. It creates the best line or decision boundary, known as a hyperplane, that can separate n-dimensional space into classes, enabling the correct classification of new data points in the future. SVM select the extreme data points that are closest to the

hyperplane, known as support vectors, which are crucial in defining the optimal decision boundary.

3.2.6 Random Forest: Random Forest is a supervised learning algorithm which is a ensemble of decision trees, where multiple trees, each of which has been trained using a randomly chosen portion of the data, are combined to give a output. The final output is created by averaging all of the decision trees' predictions, each of which makes a prediction based on the characteristic of the data. It is robust to overfitting with which a lot other models struggle.

3.2.7 K-nearest neighbours: The k-nearest neighbours (KNN) algorithm is a non-parametric, supervised learning classifier, which uses distance to make classifications or predictions about the category of a individual data point. It assigns a class to new data by choosing the K nearest data points to it. The method assumes that similar points are near each other. It is a "lazy learning" model, i.e., it only retains a training dataset and performs all calculations when making a prediction. The algorithm, during the training phase, just store the dataset, and when it gets new data, it classify that data into a category that is most similar to the new data.

3.3 AI-generated text detection:

Firstly, let’s talk about the AI-generated text percentage calculation. Here we have used two vectorization techniques TF-IDF and BERT base uncased. We have discussed about TF-IDF and BERT model earlier here BERT base uncased is the small version of BERT contains 12 encoder or Self-Attention layers has 110 million parameters which is uncased version means it generates same vector for uppercase and lowercase letters of the letters unlike BERT model which uses 24 self-attention layers. Initially TF-IDF is just used for checking as BERT base uncased model is computational costly because of 110 million parameters. The computational cost is main issue here and our dataset has *44868* unique values and take huge amount of time for creating the embedding only classification is later part. Also, the given Fig

3.3 exactly shows how BERT generates embeddings by passing through 12 encoders.

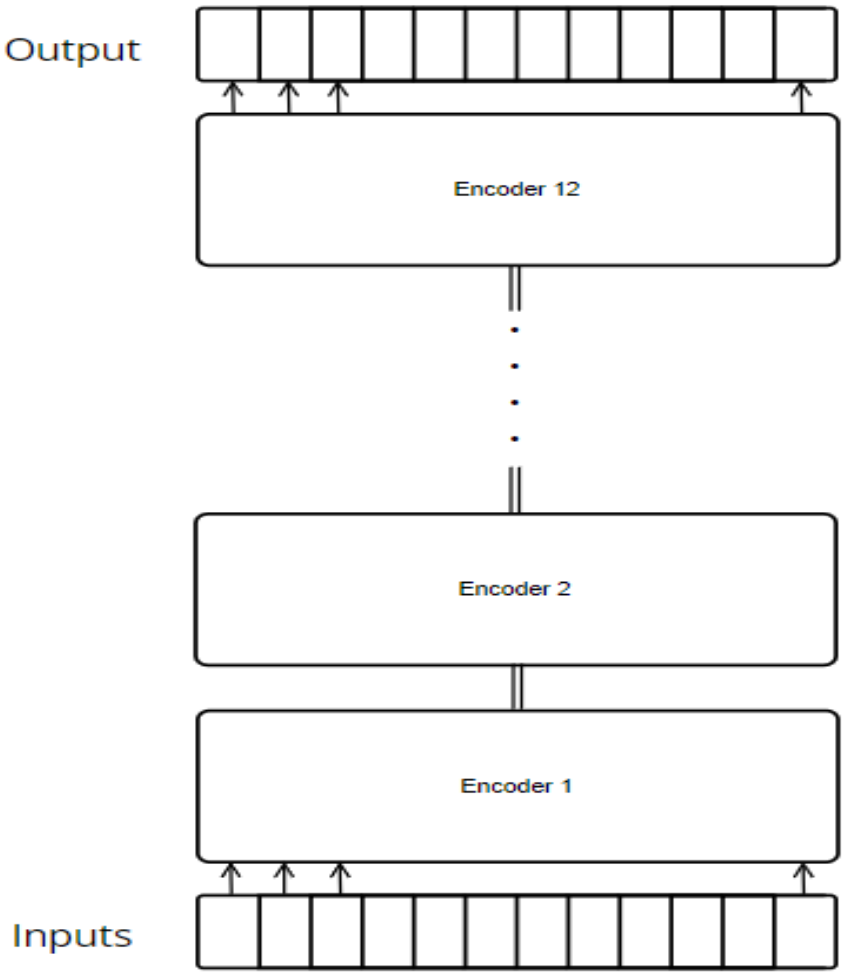

Fig 3.3 BERT Model for generating embeddings

We have our dataset so initial part is doing preprocessing of dataset converting the text into tokens and then vectorization. Let's first install the necessary python libraries we are going to use in AI detection part. First Python 3.10 version should be there as our some libraries are compatible with that version only for running the flask application. The libraries are *numpy, joblib, TfidfVectorizer, torch, BertTokenizer and BertModel* for initial steps in creating the embeddings. *Numpy* will provide the support for large, multi-dimensional arrays and matrices, along with

mathematical functions to operate on them. *Joblib* used for saving and loading our machine learning models or embeddings. *Tfidf-Vectorizer* converting text into vectors using TF-IDF vectorization technique. *Torch* is an deep learning framework that will be used for building and training neural networks with the support of GPU accelerations as we will be using BERT model for creating our embeddings. *BertTokenizer* is used for tokenizing the text and *BertModel* generating the embeddings for text analysis tasks.

The first vectorization technique is used is TfidfVectorizer() for calculating TF-IDF vectors for all the sentence in the given dataset. So, applied this technique and saved the embeddings with the help of joblib to perform prediction using the classification task. The IDF part of a paragraph is always same as total number of documents and number of documents containing term is 1.

Example:

"BERT is amazing. It is a powerful model."

Tokenised paragraph: ['BERT', 'is', 'amazing', 'It', 'is', 'a', 'powerful', 'model']

IDF = log(1 / 1 + 1) = log(1 / 2) = -0.301

And expected output will be:

BERT	amazing	is	it	Model
−0.0376	−0.0376	-0.753	−0.037	−0.037
a	powerful	model		
−0.0376	−0.0376	−0.037		

The second vectorization technique we can use is Bag of Words (BoW). So let's create their embeddings and save them with the help of joblib like TF-IDF to perform prediction using the classification task but we have to convert the numbers to float32 or float64 because certain machine learning models, especially gradient-boosted decision trees like *LightGBM and CatBoost* can only be trained on floating point format only

Example:

"BERT is amazing. It is a powerful model."

Tokenized output: ['bert', 'is', 'amazing', 'it', 'a', 'powerful', 'model']

Word Counts: [1, 2, 1, 1, 1, 1, 1] means Vector: [1.0, 2.0, 1.0, 1.0, 1.0, 1.0, 1.0]

The third vectorisation technique we are going to use is BERT Tokenizer for generating tokens of the paragraph also to separate the sentences [SEP] token is used and [CLS] token inserted at beginnings of each sentences.

Example:

"BERT is amazing. It is a powerful model. It is used for creating word embeddings."

Tokenization output from BERT Tokenizer:

[CLS] BERT is amazing . [SEP] It is a powerful model . [SEP] It is used for creating word embeddings . [SEP].

In the diagram we can see the architecture of a BERT based model in Fig 3.4.

Hence the vectorisation part is finished so we save TF-IDF, BoW and BERT embeddings also their corresponding labels using joblib library. We will be dividing the dataset into train and test part with test size of 0.2 means 20% of data will be used for testing the models and 80% data for training the models We know various about Machine Learning and Deep Learning models such as multiple layer perceptron (MLP), LightGBM, CatBoost, logistic regression, support vector machine (SVM), random forest, K-nearest neighbors (KNN) and BERT Sequence Classifier used in classification as we discussed earlier. So, we will be training every model used in classification listed above and see which one is performing well on our test dataset after getting results on test dataset we will observe its performace on real time data to see each model performace.

For TF-IDF vector embeddings we have used traditional ML and DL models like MLP, CatBoost, logistic regression, LightGBM, SVM, Random Forest and KNN with less expectations as we know it the inverse document frequency part same and

using term only for it but taking in mind that TF-IDF will be calculated for a list of assignments so that will be inverse document frequency term introduced and after the application part results of TF-IDF vectorization technique with the models calculated on train and test dataset.

To get more overview we are using various vectorisation techniques so after TF-IDF, we thinked for applying the Bag of Words technique and computing result for all these models MLP, CatBoost, logistic regression, LightGBM, SVM, Random Forest and KNN again. It is just like the previous TF-IDF technique we have used means the term part as IDF part is same still to give it a try we are using it.

Now, the main part which is BERT embeddings, we know it will be very crucial to handle this as BERT is capable of doing all these things effectively. So, to train our model we used BERT Tokenizer for tokenization of whole sentence seperating out each sentence with [SEP] and [CLS] token at the front of each document. After this as we are using BERT base model and its uncased version for generating the word embeddings and this time we are going to train one extra model which is Bert Sequence Classifier with MLP, CatBoost, logistic regression, LightGBM, SVM, Random Forest and KNN.

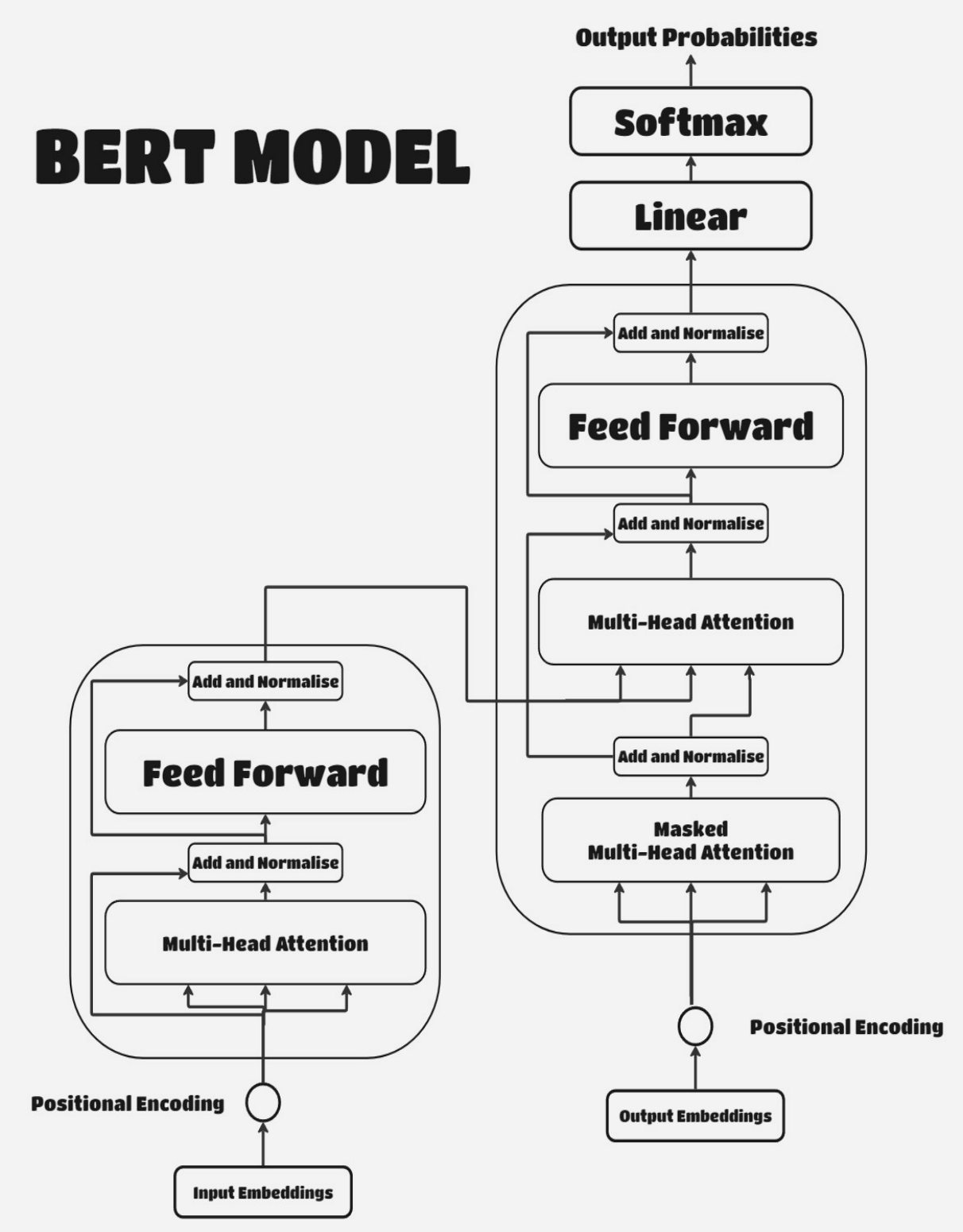

Fig 3.4 BERT Model Architecture

Let's see what are the hyperparameters in each model we are going to use for our training.

- **MLP:** We used Multi-Layer Perceptron for classifying text as human-written or AI-generated on the basis of BERT embeddings, TF-IDF and BoW. Here MLP consists of three hidden layers (128,128,64) with ReLU activations. Training of the model was conducted with 50 iterations. We saved this model for checking future real time predictions.

- **BERTSequenceClassifier:** Here also we used Bert embeddings but there are some differences in these embeddings. Here we are training a BERT sequence classification model which logits directly created for classification and it will return the probability as we will be applying softmax function. Instead of using embeddings of whole sentence it uses [CLS] token embeddings which is special token used for classification containing summary of every paragraph. Also to minimise the loss we create extra variables like best loss for remembering the loss which is minimum in 10 epoch. Keeping patience variable value as 3 so that it is stop checking for best loss if it not finding minimum value in next three epoch.

- **Other Models**: CatBoost is a gradient boosting model that can works with both numerical and categorial data discussed earlier. We have trained it using BERT embeddings, TF-IDF and BoW with 1000 total iterations, a learning rate of 0.05 and depth of 6. Applied other models too like Random Forest with 100 estimators, KNN with 3 neighbors, SVC by taking linear kernel and probability value True and logistic regression with 1000 iterations.

3.4 Similarity detection between the documents:

The similarity detection part is carried out with many vectorization techniques TF-IDF, SBERT, BoW and similarity detection with Cosine similarity and Jaccard similarity with n-gram. As shown in Fig 3.5 if angle between a and b vector is α, α

will be close to 0 if a and b are similar and 90 if a and b are not similar .

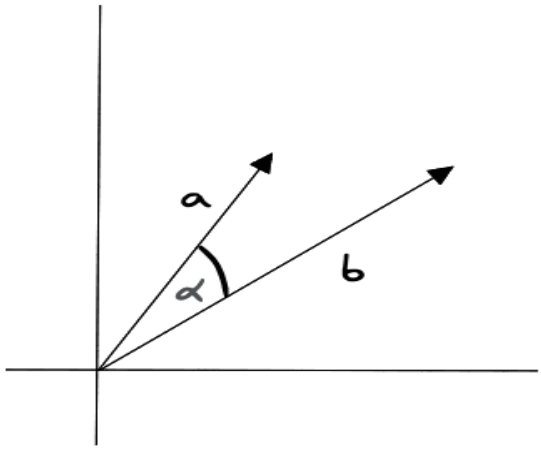

Fig 3.5 Cosine similarity of Vectors

Also the similarity score between the two vector represent their respective document is given by:

$$\text{similarity}(A,B) = \frac{A \cdot B}{||A|| \times ||B||} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}} 1$$

Jaccard similarity is calculated by two set A and B where A is set of words for Document1 and B for Document 2. It is given by the formula:

Jaccard Similarity = |A ∩ B| / |A ∪ B|

For n-gram we use n=3 i.e. trigram model and compare each text with other text for exact copy paste detection

3.5 FLASK application:

Now, results of each model will be calculated and discussed later. Let's analyse the application we have made with the help of flask, html, CSS and Javascript for making that comprehensive report. Firstly we have implemented a page where the educator can upload the zipped files which he downloaded from Google Drive containing the assignments of students. From here the process of backend begins

that to process those text files and generate output for each file. Firstly this flask application we have installed some python libraries such as *Flask* which is used to create a web app, *os* for interacting with operating system like file or directory manipulation, *zipfile* which is used to create extract and read ZIP archives. *numpy* for working with arrays, *joblib* for accessing the saved model for AI generated text prediction, cosine similarity for measuring the similarity between two vectors of documents, *PdfReader* from PyPDF2 which allows us for reading and extracting text from the PDF documents, *docx2txt* for extracting text from DOCX files, *torch* for using Pytorch to perform computation with tensors and building deep learning models such as BERT embeddings or BERT Sequence Classifier for GPU acceleration. *BertTokenizer, BertModel* just because they are going to give us best results let's compute the whole process of vectorisation from them only, *uuid* is library for generated unique IDs for object like generating unique name for files as to give educator authority to upload same file multiple times.

Let's understand the pipeline of this application. The pipeline of this flask app is firstly the uploaded file which is a zip file contains assignment done by the students either in pdf or in docx format. The flask app receives file through function request.files["file"] and extract all the files and save them to a folder "uploaded_docs" if it not exist it will make that folder. Also it creates a unique name using uuid function uuid.uuid4().hex[:8]}. Here *uuid.uuid4()* creates a random 128 bit identifier which return 8 character names for the files uploaded again and again, *.hex* convert it into hexadecimal representation, *[:8]* for slices the first 8 characters of the UUID in hexadecimal form and lastly {member} for checking whether I was a pdf or docx file. Now, for reading the file content after exracting them we are using *read_file(file_path, file_type='x')* function where it identify the file is pdf or docx. Then the file is goes under process of text extraction using PdfReader for pdf and docx2txt.process() for docx files.

From here the main part start the *predict_ai_generated(text)* is called for calculation of probability of AI generated text. The text embeddings creation is done but not like we have created earlier i.e. during training time as we are working on large

documents also. BERT has a limitation that it can train the embeddings upto a significant length which is 512 tokens. So, to handle large documents we make our methodology to split those documents into chunks of 512 tokens. Now those chunks will be going under prediction for each document and the mean probability of all the chunks give output of probability of whole document. So, *split_text_into_chunks()* is the function for it here we also applied padding as if space are pending in a chunk then the chunk will be padded with some values. After this part we perform BERT embeddings using function *create_bert_embeddings(texts)* and here the output will be the mean of all the tokens embeddings of 786 dimension i.e. a single 786 dimesional vector for a particular chunk . After that in function *predict_ai_generated(text)* we use *model.predict_proba(embeddings)* function where the model calculates the probability of chunks and at last the np.mean used for average for all the scores calculated for each chunk. This probability value is what we will be using as percentage of AI generated text.

Talking about similarity we have used Jaccard similarity with trigram model which will be helpful for exact copy paste detection. The similarities are stored in a dictionary, where each key is a tuple of document names, and the value is the cosine similarity score multiplied by 100(for percentage).

After calculating the AI-generated percentages and document similarities, results will be displayed on a web page with route *render_template("report.html", documents = document_texts.keys(), ai_scores = ai_scores, similarity_scores = similarity_scores)*. This page will be showing the list of document names, AI generated probability as percentage and Similarity scores between documents. The document which has very high AI percentage means above 75% will be shown in dark red color, high (between 50% and 75%) in red color, medium (between 20% and 50%) in yellow and Low (up to 20%) will be shown in green color. The computation time might be more but it the most detailed and exact AI and similarity percentage calculation.

3.5.1 Workflow Diagram:

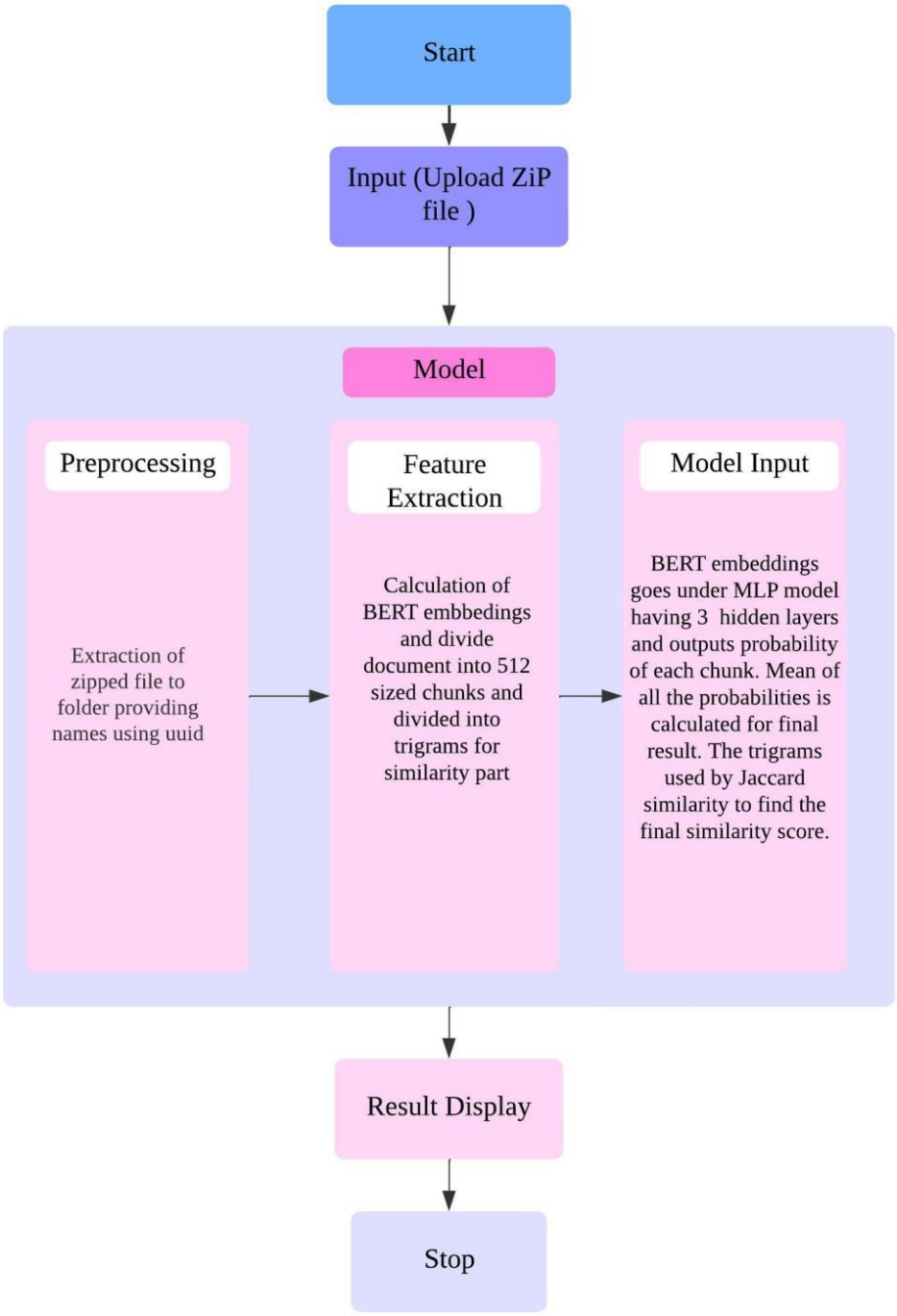

Fig 3.6 Workflow Diagram

4. Experimental Setup

In this section, we discuss the setup and configurations we required for implementing this project:

4.1 Dataset:

Data collection is very important in every machine learning task so we have to choose a dataset which contains large amount of data written by human and generated by AI and also the AI text should be generated not only from one AI model only as many AI tools are available in the market so we have to work on data generated by all the AI models.

Training Data: Training data for AI detection was compiled from samples of AI-generated and human-written content i.e. available DAIGT dataset available on Kaggle, while similarity detection training involved a dataset of provided documents only.

Real-Time Data: Testing was conducted on simulated real-time submissions to evaluate model adaptability. We use ChatGPT in calculation of real time AI generated percentage detection. Also for human written text we have used our previous semester reports for comparison. Also gathered the assignments of assignments from my own classmates through Google Drive link.

4.2 Environment:

The project was developed in a Python-based environment using libraries like scikit-learn, joblib, CatBoost, PyTorch, and transformers. The MLP model was deployed as a primary classifier, while Flask served as the web application framework integrated with HTML, CSS and Javascript for frontend design and interactivity.

4.3 Technology Stack:

In backend part we have used Python and Flask. The text preprocessing part with BERT embeddings, TF-IDF and cosine similarity, At last, frontend we used HTML, CSS and Javascript integrated with Flask.

5. Results and Analysis

In this section, we are going to see the results which are obtained from various AI models and not only that we are also going to see how our model is performing on real time data.

5.1 AI Detection Models:

Here, we discuss the output generated by each AI model on the basis of all the vectorization techniques we have followed:

5.1.1 BERT Embeddings:

Firstly, let's calculate the output generated by each AI model on the basis of BERT embeddings:

Multi Layer Perceptron(MLP): In this model we have achieved high accuracy on of 99.54% which is good but for such a high accuracy our data can be overfitted on dataset only so it is very crucial to check it real time also. Checking it on real time data we got excellent result and this is going to be our model for prediction in main flask application.

LightGBM and CatBoost: On testing the data from the dataset we get high accuracy of 99.15% and 99.31% respectively. They also yield good accuracy for real time data although LightGBM has good accuracy for human written documents on the other hand CatBoost performed well on the AI generated text.

Logistic Regression and SVM: SVM and Logistic Regression gives an accuracy of 99.4% and 99.4% respectively. Both predicting average on real time AI generated text data on observation.

Random Forest and KNN: Both models gives great result on test dataset of 98.99% and 99.33% but according to the observation on real time data they are performing extremely poor.

Bert Sequence Classifier: We got high accuracy of 99.61% on our dataset with minimum loss as 0.0007056075760007724 on 3rd epoch using Adam Optimizer.

But on real time data it doesn't work well means model is overfitted it may be due to chunking of texts as BERT Sequence classifier works only on the basis of [CLS] token and while dividing the text into the chunks, the [CLS] token will be added in the front of the chunk where the context may get loosed for the sentence. So, it is not preferred.

The given table represents the test accuracy on the dataset given by the above discussed models:

MODEL NAME	TEST ACCURACY
MLP	99.54
LightGBM	99.15
Catboost	99.31
Logistic Regression	99.40
SVC	99.45
Random Forest	98.99
KNN	99.33
BERT Sequence Classifier	99.61

Table 5.1 Output using BERT

From the study and observation, it is decided that Multi Layer Perceptron(MLP) is

best model for analysing the real time data it get low percentage on human written assignment whereas gives high percentage on AI-generated assignments.

5.1.2 TF-IDF vectorization:

Here also we get all the model performing very well and seems well trained on train and text dataset of all the models. Talking about real time, it lags and misinterpret the percentage value even shows a human written But its limitations is that it doesn't preserve the context of sentence which make it unreliable for using for real time data. There is example that how it can preserve the context of word:

Sentence 1:

"The bank is on the river."

Here, "*bank"* refers to the side of a river.

Sentence 2:

"I deposited money in the bank."

Here, "*bank"* refers to a financial institution.

It just use formula and see this word is same and gives TF-IDF score for that. Here in both the sentences talking them into different documents we get they are having same TF-IDF score. That's why we can't use TF-IDF on real time data for presenting the authentic report.

Here is the train and testing accuracy given by TF-IDF vectorization technique in *table 5.2.*

As expected, the BoW model too performed good after observing the results given by TF-IDF vectorization technique on our dataset only with accuracies near 98% for test. But perform poor for real time data. So this BoW technique is also declined as per our observation.

5.2 Similarity Detection Performance:

Let's discuss the best vectorization and similarity score technique observed:

5.2.1 Cosine Similarity:

It is best for detecting whether two vectors are especially in context of text or document similarity, because it handles every aspect that if two particular vectors are identical means their angle would be zero gives high similarity score and if not

identical their cosine score coming closer to one means the angle between them is 90 i.e. low similarity score. But it may give wrong results for large documents i.e. where the size of the vector corresponding to document is very large.

MODEL NAME	TRAIN ACCURACY	TEST ACCURACY
MLP	100	99.33
LightGBM	99.85	98.96
CatBoost	99.94	99.08
Logistic Regression	99.29	99.13
SVC	99.71	99.48
Random Forest	100	98.34
KNN	99.36	98.43

Table 5.2 Output using TF-IDF

5.2.2 Jaccard Similarity:

As we know it requires no vectors for finding the similarity as it only rely on the ratio of the same number of words between two documents to the vocabulary size of both the documents which fails in identifying the exact copy pasting of a particular word.

To address this we used n-gram by taking n=3 i.e. trigram model in calculating it to get best results.

5.2.3 TF-IDF with Cosine Similarity:

After vectorization by TF-IDF, we successfully detected direct copy-pasting of documents and got an excellent technique to detect similarity using cosine

similarity. We used this as it provides us high uniqueness of the text from all the documents. Common words like a, an, the, in are not rare and used frequently whereas rare words will be having high TF-IDF score. Now, the computed TF-IDF vectors of both documents goes for similarity detection where we are using the cosine similarity and a particular rare word is used in one document only rather than the other document then it will give less similarity score but it will be high if other student also used same word. But it may give wrong results for large documents i.e. where the size of the vector corresponding to document is very large. So, using it for larger documents is not an idle method.

5.2.4 SBERT:

Since we knows the capability of BERT model from our AI detection part, here SBERT will not be as useful as it also preserve the context of answer and we know a particular question of assignment is done by a student, it will carry a contextual meaning but main issue is that questions can have similar answer we can change the way of delivering the answer but not the context of the answer which results high similarity score. That's why we are not using SBERT model in similarity detection part for creating the vectors.

5.2.5 BoW:

It is good to use this technique for vectorization but this has a limitation that it can give score of cosine value greater, for this we can do normalization but it won't help it gives high similarity scores only while using with cosine similarity. So, it is good to drop this idea for vectorization.

5.3 FLASK Application Output:

The output we get after uploading zipped containing the assignments of students:

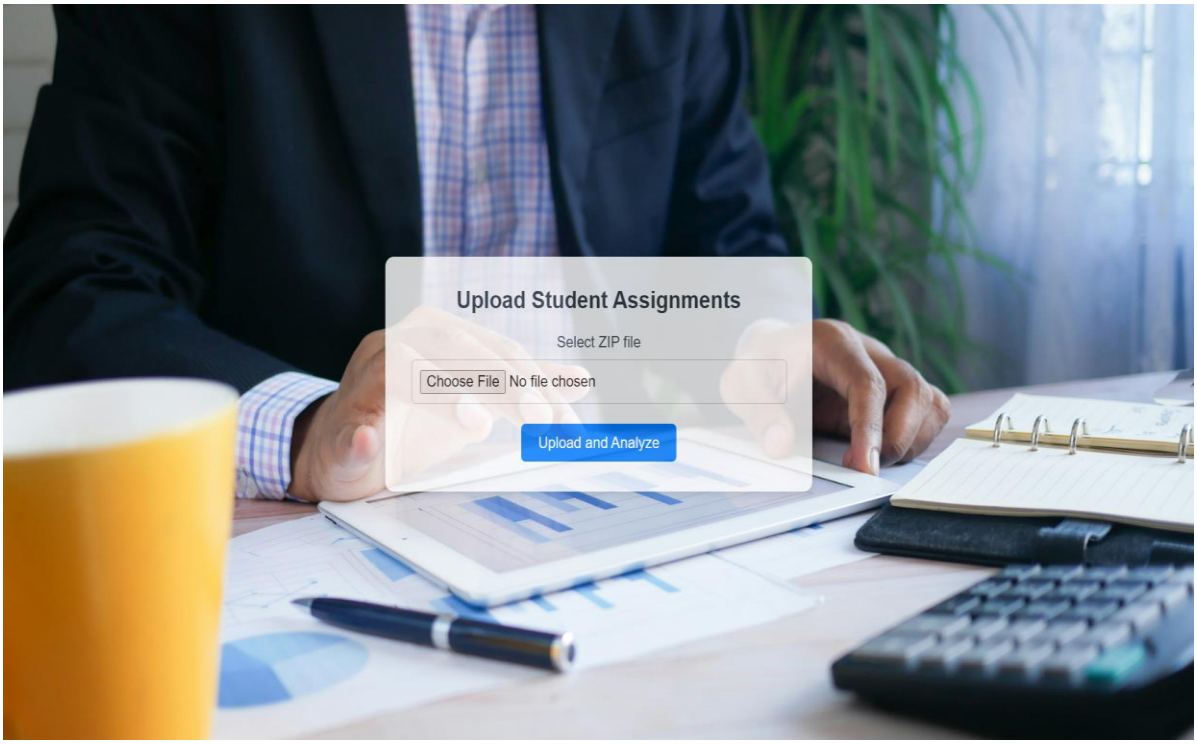

Fig 5.1 Flask Interface Input

After uploading the document this is how our report will look like:

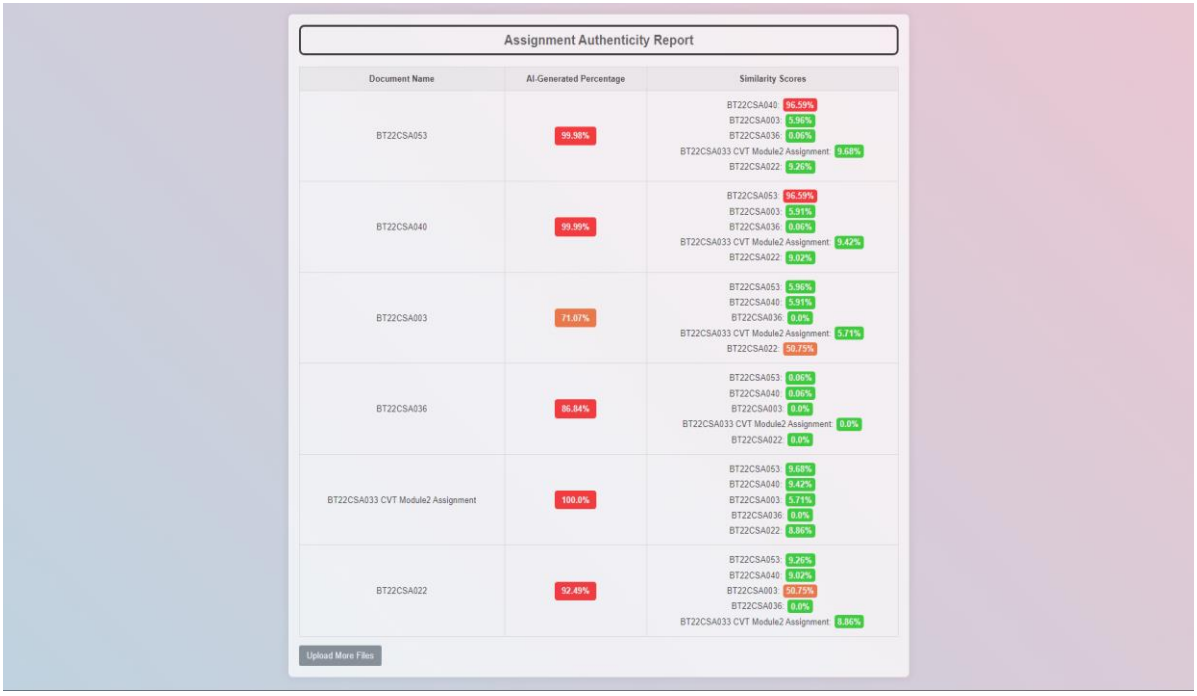

Fig 5.2 Flask Interface Output

Conclusion and Summary

In conclusion, we have found the best embeddings are BERT embeddings for detecting the AI generated text rather than simple vectorization techniques due to the context preserving feature provided by it and we combined BERT embeddings with MLP for finding out the probability of each text. The value of probability we are getting is declared as percentage of AI generated text. There is no limit or restriction of words in our model. We also used other vectorization techniques like TF-IDF and Bag of Words too but as they don't generate context based embeddings and also not performed well on real time data we dropped those techniques.

In similarity detection part, we concluded that Jaccard similarity with trigram produces good result in exact copy paste detection work. We have also used Bag of Words and Sentence BERT in detecting the similarity score but it cannot be practical for using them for exact copy paste detection.

Finally, in Flask application educator will upload zipped folder containing the assignment given by students. We applied chunk based BERT embeddings for vectorization of text and MLP for predicting the AI generated probability. For similarity score Jaccard similarity is used with trigrams for every documents and report get generated.

References

1. Mohamed A. El-Rashidy, Ramy G. Mohamed, Nawal A. El-Fishawy, Marwa A. Shouman, *An effective text plagiarism detection system based on feature selection and SVM techniques*, International Journal of Research Publication and Reviews, 2024.
2. Yuhong Mo, Hao Qin, Yushan Dong, Ziyi Zhu, Zhenglin Li, *Large Language Model AI text generation detection based on transformer deep learning algorithm*, 2024.
3. Shanmathi V1 , Sowmiya K1 , Aishwarya SH1 , Sakthivel M2, *Document Plagiarism Checker Based on Similarity*, 2024.
4. Galal, O., Abdel-Gawad, A.H. & Farouk, M. *Rethinking of BERT sentence embedding for text classification,Neural Comput & Applic* **36**, 20245–20258 (2024).
5. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin *Attention is All you Need*, 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.
6. Tri Puspa Rinjenia , Ade Indriawana , Nur Aini Rakhmawatia* *Matching Scientific Article Titles using Cosine Similarity and Jaccard Similarity Algorithm* Seventh Information Systems International Conference (ISICO 2023), 2024
7. Alvi Faisal, Stevenson Mark, Clough Paul (2021) *Paraphrase type identification for plagiarism detection using contexts and word embeddings*, Int J Educ Technol Higher Educ 18(1):1–25 International Journal of Educational Technology in Higher Education
8. Amil Merchant, Elahe Rahimtoroghi, Ellie Pavlick, Ian Tenney *What Happens To BERT Embeddings During Fine-tuning*, 2020
9. Bushra Alhijawia,∗ , Rawan Jarrara , Aseel AbuAlRuba , Arwa Badera *Deep Learning Detection Method for Large Language Models-Generated*

Scientific Content, Princess Sumaya University for Technology, Amman, Jordan, Feb 2024.

10. Abnar S, Dehghani M, Zamani H, Shakery A (2014) *Expanded N-Grams for Semantic Text Alignment*. In: CLEF (working notes) 1180:928-938.
11. H.Zhang, M.Huang and W. Li," .Research on text similarity measurement hybrid algorithm with term semantic information and TF-IDF method "-2022.
12. QuillBot: *https://quillbot.com/ai-content-detector*
13. GPTZero (2023). *https://gptzero.me/* access date: 27/8/2023.