

# Semantic Plagiarism Checker on Handwritten Scripts



**INDIAN INSTITUTE OF TECHNOLOGY JODHPUR**

Trimester II Winter 2020

**Guided By**

**Dr. Romi Banerjee**

**and**

**Dr. Debarati Bhunia Chakraborty**

**Submitted By :**

**Aditya Kumar  
B18CSE002**

**Kartik Vyas  
B18CSE020**



# Table of Contents

1 Overview	3
2 Motivation	4
3 Goals	5
4 Problem Statement	6
5 Datasets	7
6 Development Plans and Execution	10
7 Website Integration	30
8 User Guidelines	32
9 Limitations and Future Plans	33
10 Result	33
11 Techstack	33
12 Demo and Github Link	33
13 Conclusion	34
14 Acknowledgments	34
15 References	35

# 1 Overview

Letters, words, phrases, sentences, paragraphs, scripts, documents form a very fundamental and crucial part of our lives. There is an abundant amount of textual data and our life is deeply governed by how we perceive it. In the lives of students, there are countless number of submissions that they make in the format of quizzes, assignments, projects, exams, etc. It then falls upon the duty of instructor to evaluate and grade all this large amount of data.

It is to be noted that just evaluation is not sufficient, there has to be a lot of factors that needs to be looked into, such as the semantics of the text, whether it is in the same context as to what is desired, whether it falls under entailment or contradiction category with respect to the desired answer. Another crucial aspect is whether or not the thoughts/ideas/work is original; that is, whether the work is plagiarised or not. To account for that we must understand what we define plagiarism as.

Plagiarism is the representation of someone else's work or ideas as our own work and it can be equal to a crime. Plagiarism is growing day by day. With a huge set of information on the internet, one can easily get any information and claim it as their original work and get away with plagiarism. This is not it, there can be cases of plagiarism involved through cheating as well.

On another note, checking plagiarism/similarity using any pre-existing tool is not feasible due to the format of submission. The format can be a pdf of scanned images or a pdf of digital text or simply a text file; all these formats needed to be handled individually and then brought down to a common format.

Thus, there are a lot of open problems lying around in the world with no apparent single solution for checking various kinds of plagiarism, semantic similarity, textual entailment analysis and text overall analysis. We have developed a solution that answers all these questions and this tool can aid the instructors and students to improve the quality of education and this tool has various real life applications in other fields as well.

## 2 Motivation

More and more number of Educational Institutes are now stressing upon the importance of continuous evaluation. This directly impacts the number of assessments to be done for various projects, quizzes, assignments and exams in general. It becomes a time and resource intensive task for the instructors to evaluate all the submissions with remarkable accuracy. Additionally, the instructor also needs to keep a check whether any student has plagiarized from the web or there is any case of intrinsic plagiarism among the students. Then, how similar the actual submission of the student is with reference to what is expected from the student.

There are tools available for such requirements but almost all of them are paid and moreover, instructors need to run them one by one, which can be a pretty difficult task in itself. Thus, we developed a one-stop solution for the entire problem. Consider that there are  $n$  number of students submitting one document each. Each of these documents would be checked against each other for plagiarism and then each document would also be checked against all web relevant searches for plagiarism. Moreover, every pair of the sentence would be checked for how much semantic similarity is there between them and then their textual entailment would also be done, denoting whether they fall under the category of entailment, neutral or contradiction.

Additionally the format of all the submissions should be brought down to a single format as a preprocessing step so data can be analysed further accordingly. We worked upon how to convert PDFs both digital text and handwritten text and then to convert them a .txt file with uniform digital text.

This would help the instructors in not just checking for plagiarism but actually easy evaluation of the submissions, which would be a relative analysis and even the whole grading can be automated according to the instructors' use case. The single thought of freeing the instructors' precious time which can be used in useful teaching and overall improving of the education experience motivated us to pursue the project throughout the course of the project. It is to be noted that it is not just for educational purposes, this tool can be used to

analyse already published and yet to be published research papers for proper analysis of the topic and the research papers.

## 3 Goals

As outlined by the previous two sections, the goal is to develop a system that helps in first bringing all the submissions in a single format and then consider checking plagiarism on various scales and then doing semantic analysis by predicting a similarity score and recognizing textual entailment.

We plan to do this with the help of diverse and deeply resourceful and integrated Python libraries and building PyTorch and TensorFlow model Binary Classification, Regression, and Multi-Class models. We have used several datasets to validate the work and train the models. We thought of a fresh approach and decided that instead of focusing on one type of feature extraction/embeddings, we would form a list of arrays and then decide what approach works the best for our work. The following are briefly the goals that we focused on during the course of the project, we have finalised the best techniques for the below.

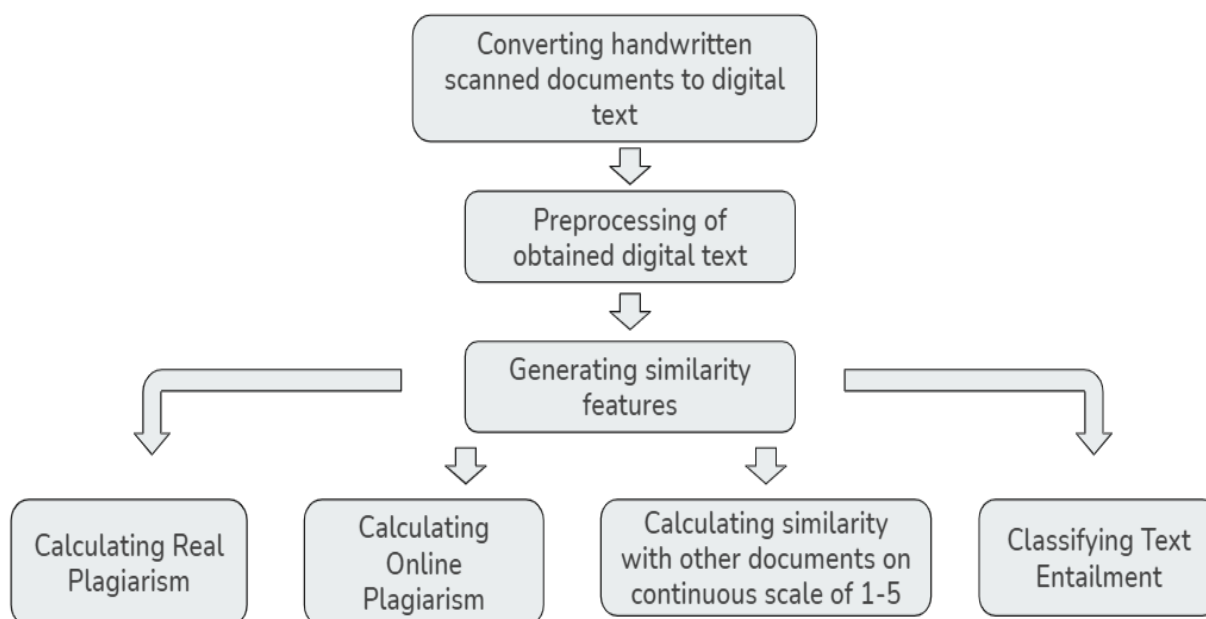
- ❖ Convert all formats into single .txt format
  - Convert Scanned handwritten documents pdf into digital text
  - Convert digital pdf to .txt format
- ❖ Compute extent of online plagiarism
- ❖ Compute extent of plagiarism among documents [all pairs possible]
- ❖ Compute features that would aid in getting idea of similarity/plagiarism
  - Cosine Similarity
  - Cosine Trigram Similarity
  - Jaccard Similarity
  - Longest Common Subsequence
  - Sequence Matcher

- Universal Sentence Encoder
  - BERT Sentence Embeddings
  - Word2Vec Spacy Sentence Embeddings
  - Rabin Karp
  - NLTK Phrase Tool
  - NLTK Docism Took
  - N-gram Containment Tool
- ❖ Compute extent of semantic similarity among documents [all pairs possible]
  - ❖ Recognizing textual entailment among documents [all pairs possible]

We have worked upon and successfully implemented the above mentioned goals, details of which are later in the sections.

## 4 Problem Statement

We would be given a set of documents in varied formats such as pdf files of scanned handwriting, pdf files of digital text and .txt files. The aim is to analyse the set of documents thoroughly with the tools, techniques and features as mentioned above in the Goals section to help out the instructor / any body that wants to analyse a set of documents, research papers, informative articles, etc. We would have the idea about web plagiarism, plagiarism in between documents, similarity in between documents and textual entailment analysis among all pairs possible among the documents.




## 5 Datasets

### 5.1 AI dataset

During Trisem1 2020-21 in Artificial Intelligence (CS323 course), CSE and BSBE batch submitted class notes in two different formats, either handwritten or digital format. These notes are a great dataset for our project as this is real-world data. We floated a form to collect those notes from our fellow batchmates. Around 35 students gave their notes along with the consent to use their notes as a dataset. Each student made four submissions and on an average, each submission was 2500 words long. This helped us in properly analysing the semantics as every student explained the same concept in their own words. We also gathered the data whether the students' work was penalized or not as a part of the coursework and then judged the accuracy of our tools accordingly.

### 5.2 DBMS and ADA Notes

These notes were provided by Dr. Romi Banerjee which were written during her B. Tech days and currently are used as reference notes by the students.



These notes are a great source of information in various different formats, using different types of diagrams and notations. There is variation in the ink colour, thus this gives us the variety to judge the accuracy of Microsoft API. We converted the pdf into images, where one image has one page of the pdf, then we extracted and recognized the text in each image using API call and this text got saved into a .txt file, which we later converted into a pdf file. Thus, the entire 80 pages notes were converted into a pdf file in digital text format.

### 5.3 MSRPC Dataset


This dataset<sup>[1]</sup> is provided by Microsoft and is known as Microsoft Research Paraphrase Corpus. It consists of 5800 pairs of sentences. Along with a label to indicate whether a pair of sentences captures a paraphrase/semantic equivalence relationship or not. This is a binary classification measure of such an equivalence.

### 5.4 SICK Dataset

It consists of 10,000 pairs of sentences along with a sentence relatedness score and a label to indicate entailment relation between two sentences. Sentence relatedness score is labeled on a 1 - 5 rating scale. While for entailment relation the sentences are labeled in three different classes “contradiction”, “entailment”, and “neutral”. The distribution of the dataset<sup>[2]</sup> for sentence relatedness score is as follows, 923 pairs within the [1,2) range, 1373 pairs within the [2,3) range, 3872 pairs within the [3,4) range, and 3672 pairs within the [4,5] range. The distribution of the dataset for entailment relation is as follows, the entailment annotation led to 5595 *neutral* pairs, 1424 *contradiction* pairs, and 2821 *entailment* pairs. This dataset is highly beneficial for us as it gives a measure of semantic similarity on a continuous scale and additionally, gives us the dataset to analyse whether two scripts are related in terms of entailment or not. A complete analysis leads us to better understanding and this can be used to judge students’ responses and possible grade them.

### 5.5 STS Dataset





This dataset<sup>[3]</sup> considers the semantic similarity of nearly 8000 independent pairs of texts and shares a precise similarity metric definition of assigning a number between 1 to 5 to each pair denoting the level of similarity/entailment. The dataset is inspired from different genres like news, forums, and captions; which makes it diverse. We have used this to check how the model architecture works on a different kind of data. This helped us in analysing and fine tuning, thus improving the model and the overall accuracy.

## 5.6 University of Sheffield

This dataset<sup>[4]</sup> is provided by the University of Sheffield. 19 students volunteered for making the dataset. There were five different questions related to computer science. For source text, the answer to the question was taken from Wikipedia. 19 other students gave their answers to respective questions which resulted in 95 answers. In all, there were 100 documents of which 95 were answers provided by students and 5 were source documents to which answers were compared for plagiarism.

There were four levels of plagiarism :

**Near Copy:** Participants were asked to answer the question by simply copying from a relevant Wikipedia article.

**Light Revision:** Participants were asked to answer the question on the basis of the Wikipedia article along with altering texts using some basic ways like substituting phrases and words with synonyms.

**Heavy Revision:** Participants were asked to answer the question on the basis of Wikipedia article but this time they were asked to alter the text heavily by replacing phrases and words with synonyms, mixing sentences, or splitting one sentence into many sentences:

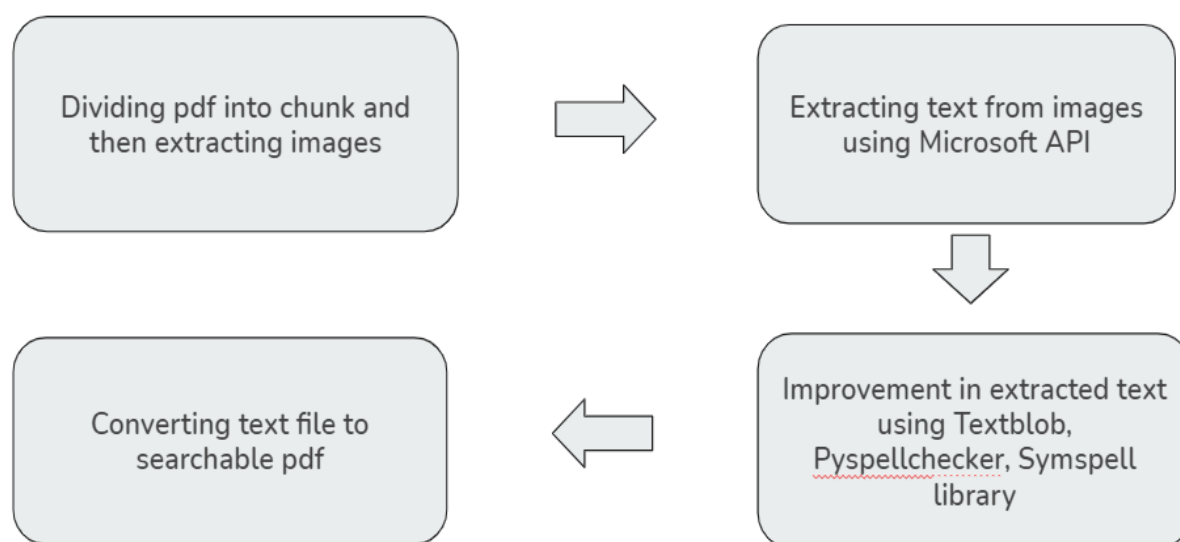
**Non Plagiarism:** In this case participants were given lecture notes or sections from the textbook and were asked to answer the questions after reading the material.

This dataset represents the type of plagiarism practiced by students in academic settings and has been of relevance for us to figure out the accuracy,

quality and usefulness, and use case of each feature that we have used to compute the similarity score.

## 6 Development Plans and Execution

### 6.1 Converting Handwritten Scanned Documents into Digital Text



#### 6.1.1 Extracting images out of PDF

The pdf is divided into smaller chunks and then each chunk is processed separately. RAM is flushed after processing of each chunk to increase speed. Images are extracted from pdf chunks using pdf2image along with poppler library.

#### 6.1.2 Extracting text out of images using Microsoft API

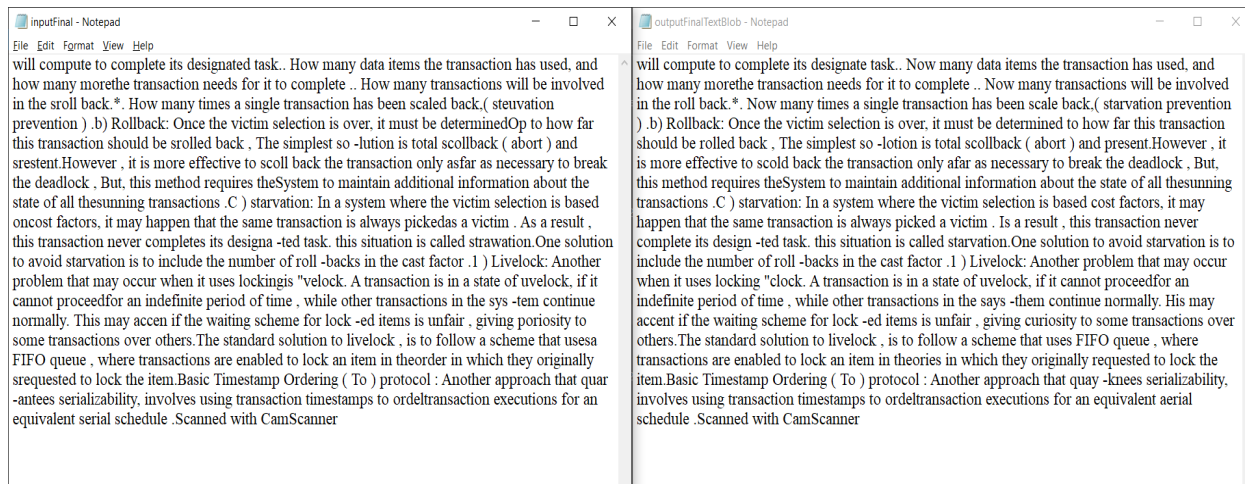
Then the text is extracted from images using Microsoft API<sup>[5]</sup> and is being appended to a text file. We get pretty good results but it needs further improvements.

### 6.1.3 Improvement in the extracted text

The text obtained from API is further improved using different libraries like textblob, Pyspellchecker, and Symspell library.

#### 6.1.3.1 TextBlob Library :

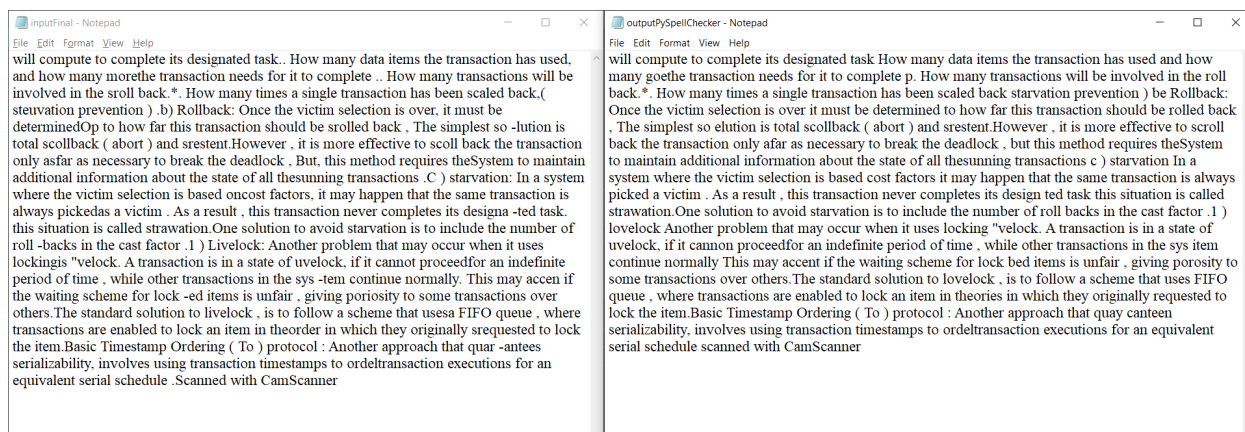
It is based on Peter Norvig's "How to Write a Spelling Corrector". The code generates two different sets of words for the incorrect word. The first set contains the words that are generated by making only one edit in the incorrect word while the other sets of words are generated by making two edits in the incorrect word. These words are further narrowed down by checking their existence in the dictionary. The final word to be replaced is chosen with the help of probability. The probability of the candidate word appearing as a word in English Text and the probability that given incorrect word would have been typed when the author wanted to write the candidate word. A combination of the above two probabilities is taken into consideration while choosing the candidate word.



#### 6.1.3.2 Pyspellchecker Library:

It is also based on Peter Norvig's "How to Write a Spelling Corrector". It uses the Damerau Levenshtein distance algorithm to find permutations with an edit

distance of 2 from the original word. It then compares all the permutations generated by insertions, deletions, replacements, and transpositions to known words in a word frequency list. The words that are found more often in the frequency list are more likely the correct words for replacing the current word.



### 6.1.3.2 Symspell Library :

The Symmetric Delete spelling correction algorithm algorithms<sup>[6]</sup> reduces the complexity of edit candidate generation and dictionary lookup for a given Damerau-Levenshtein distance. It is six orders of magnitude faster (than the standard approach with deletes + transposes + replaces + inserts) and language independent. Opposite to other algorithms, only deletes are required, no transposes, replaces and inserts are required as transposes, replaces and inserts of the input term are transformed into deletes of the dictionary term. Replaces and inserts are expensive and language dependent: e.g. Chinese has 70,000 Unicode Han characters!. The speed comes from the inexpensive delete-only edit candidate generation and the pre-calculation. An average 5 letter word has about 3 million possible spelling errors within a maximum edit distance of 3, but SymSpell needs to generate only 25 deletes to cover them all, both at pre-calculation and at lookup time.

The SymSpell algorithm exploits the fact that the edit distance between two terms is symmetrical so we can combine both and meet in the middle, by transforming the correct dictionary terms to erroneous strings, and transforming the erroneous input term to the correct strings. Because adding a

char on the dictionary is equivalent to removing a char from the input string and vice versa, we can on both sides restrict the transformation to deletes only.

## Single word spelling correction

Lookup provides a very fast spelling correction of single words.

- A Verbosity parameter allows to control the number of returned results:  
Top: Top suggestion with the highest term frequency of the suggestions of smallest edit distance found.  
Closest: All suggestions of smallest edit distance found, suggestions ordered by term frequency.  
All: All suggestions within maxEditDistance, suggestions ordered by edit distance, then by term frequency.
- The Maximum edit distance parameter controls up to which edit distance words from the dictionary should be treated as suggestions.
- The required Word frequency dictionary can either be directly loaded from text files (LoadDictionary) or generated from a large text corpus (CreateDictionary).

## Compound aware multi-word spelling correction

LookupCompound supports compound aware automatic spelling correction of multi-word input strings.

### 1. Compound splitting & decompounding

Lookup() assumes every input string as *single term*. LookupCompound also supports *compound splitting / decompounding* with three cases:

1. mistakenly inserted space within a correct word led to two incorrect terms
2. mistakenly omitted space between two correct words led to one incorrect combined term
3. multiple input terms with/without spelling errors

Splitting errors, concatenation errors, substitution errors, transposition errors, deletion errors and insertion errors can be mixed within the same word.

## 2. Automatic spelling correction

- Large document collections make manual correction infeasible and require unsupervised, fully-automatic spelling correction.
- In conventional spelling correction of a single token, the user is presented with multiple spelling correction suggestions.  
For automatic spelling correction of long multi-word text the algorithm itself has to make an educated choice.

Examples:

- whereis th elove hehad dated forlmuch of thepast who couqdn'tread in sixthgrade and ins pired him

+ where is the love he had dated for much of the past who couldn't read in sixth grade and inspired him (9 edits)

- in te dthird qarter oflast year he hadlearned ofca sekretplan

+ in the third quarter of last year he had learned of a secret plan (9 edits)

### Word Segmentation of noisy text

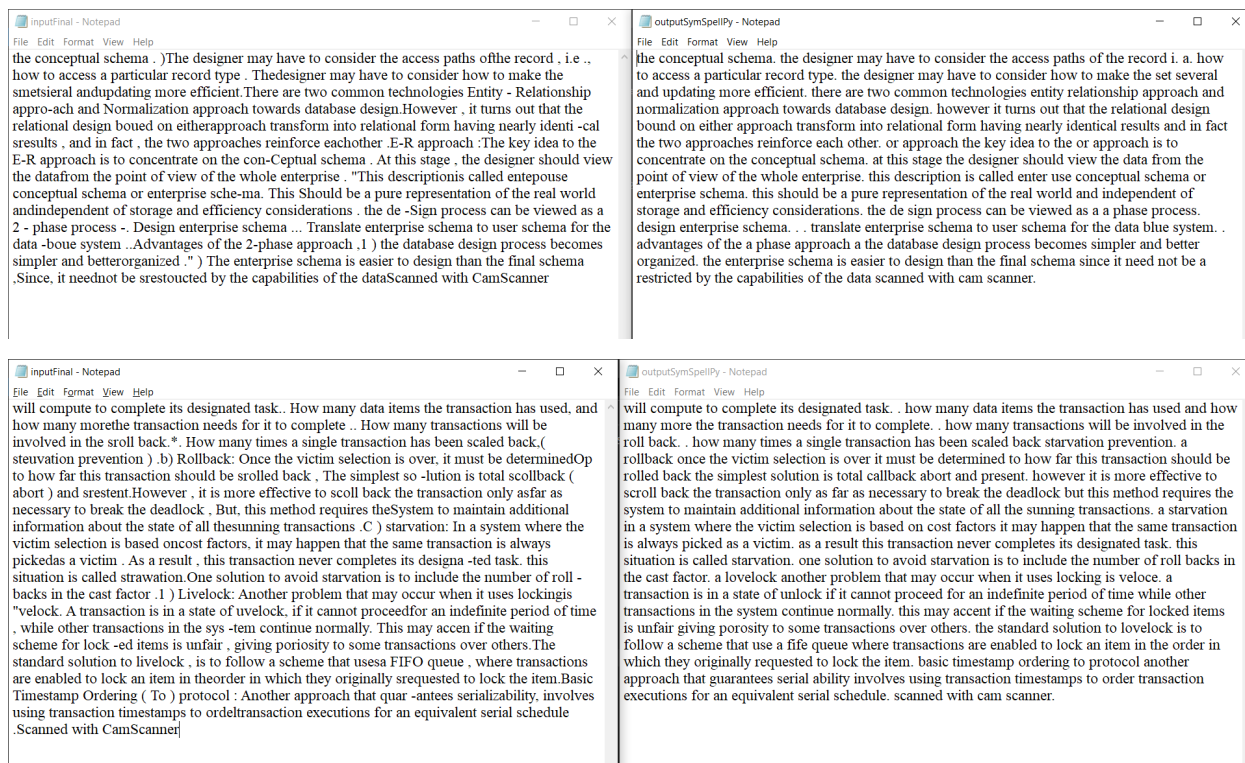
WordSegmentation divides a string into words by inserting missing spaces at appropriate positions.

- Misspelled words are corrected and do not prevent segmentation.
- Existing spaces are allowed and considered for optimum segmentation.
- SymSpell.WordSegmentation uses a Triangular Matrix approach instead of the conventional Dynamic Programming: It uses an array instead of a dictionary for memoization, loops instead of recursion and incrementally optimizes prefix strings instead of remainder strings.
- The Triangular Matrix approach is faster than the Dynamic Programming approach. It has a lower memory consumption, better scaling (constant  $O(1)$  memory consumption vs. linear  $O(n)$ ) and is GC friendly.
- While each string of length  $n$  can be segmented into  $2^{n-1}$  possible compositions,

SymSpell.WordSegmentation has a linear runtime  $O(n)$  to find the optimum composition.

Examples:

- thequickbrownfoxjumpsoverthelazydog
- + the quick brown fox jumps over the lazy dog



## 6.1.4 Conversion of the text file to PDF


Finally, the text file is converted to a searchable pdf using the txt2pdf library.

## 6.2 Features chosen as Measures

### 6.2.1 Bert Sentence Encoder

BERT<sup>[7]</sup> stands for Bi-Directional Encoder Representational from Transformers. For sentence encodings, we map a variable length input text to a fixed sized dense vector. We feed in the input to the BERT, followed by a pooling layer to





ensure that the vector is of equal dimensions. The embeddings for the BERT layer consists of three basic components : token, segment, and position embeddings. The token embeddings are basically the pre-trained word embeddings developed from a vocabulary of 30K tokens, the segment embedding refers to the sentence number that is embedded in the vector and the position embedding refers to the position of word within a phrase that is encoded; as the model contains no recurrence and no convolution, for the model to make use of the order of the sequence, it must inject some information about the relative or absolute position of the tokens in the sequence, that is what the "positional encodings" does. The output of the BERT layer are the word vectors which are then passed from a Softmax activated dense layer where loss is computed focusing mainly on the masked tokens to raise the context analysis. This model is much better than LSTM as it focuses on each word in parallel rather than just the forward and backward direction of sentences and is much faster as well than LSTM.

The overall idea is that words/phrases with similar meaning lie closer to each other in the vector space. The encoder block broadly consists of two parts : multi-head attention mechanism and a simple position wise fully connected feedforward mechanism; the attention component focuses on the importance of a word in a sentence and its role in the sentence and the feed forward unit transforms the vectors in such a format that can be processed further by the next encoder block.

Basically, using Google's BERT for encoding two documents into vectors and then finding cosine similarity between two vectors.

### 6.2.2 Cosine [1]

In this feature, we use the technique of first converting the input paragraph into a vector. To make this vector, we use the frequency of each word and represent it in vector format. Thus, in this way the order of words does not matter and just the frequency does. Then, we calculate the inner product of these two vectors and thus, compute the similarity score.

### 6.2.3 Cosine [2]

In this feature the technique is the same as stated above, however the way sentences are encoded in vectors is what makes the clear difference. First, we



tokenize the sentences into words and then we remove all the stop words such as [the, a, an, he, she, etc]. Then, two vectors are formed of size equal to the union of the token words. One vector represents one of the input and the corresponding value to the word token is binary, either 0 or 1 depending upon the presence of word in that input. Then, the inner product of these two vectors is taken and the similarity score is assigned.

## 6.2.4 Cosine Trigram

We first remove the punctuation and the stop words of the input phrases and compute all the possible trigram from the two phrases. Then, we calculate term frequency, document frequency, and inverse document frequency for both documents. Document frequency for each term was calculated by counting the number of documents that had the given term.

The formula for Inverse Document Frequency:

$$IDF(t) = 1 + \log N / DF(t)$$

Where

DF(t) : Document frequency for term t

IDF(t) : Inverse Document Frequency for term t

N: Total number of documents

Here it is to be noted that the term can be unigram, bigram or trigram. In our case, we went with trigram to focus more on the context and ordering of words rather than just the presence of words. Then weighted vectors are computed using values of term frequency and inverse document frequency for both documents. Then the dot product is calculated between TF-IDF vectors.

## 6.2.5 Docsim nltk

The similarity formula returns a high value when the content of the test document is either a subset or a superset of the registered document. It works on a set of words that are in common between test and registered documents. The value of plagiarism is calculated using the concept of scam distance. It is a relative measure to detect overlap, irrespective of the differences in the document size.

The formula for so is:

$$S(T, R) = \frac{\sum_{w_i \text{ belonging to } C} (f_i(R) \times f_i(T))}{\sum_{i=0}^N (f_i(T) \times f_i(T))}$$

Where

$f_i(R)$  and  $f_i(T)$  are the number of times  $w_i$  occurs in registered document R and test document T.

The above formula returns the degree to which R overlaps T, normalized with the document T alone. The relative similarity between documents is given by

$$\text{Similarity}(T, R) = \max(S(T, R), S(R, T))$$

Where  $S(R, T)$  is the same formula just as above with reversed operands.

## 6.2.6 Word2Vec Spacy Embedding

Word2Vec utilizes vector representations of words, "word embeddings". Word embedding is a popular framework to represent each word as a vector in a vector space of many (for example 300) dimensions, based on the semantic context in which the word is found. This technique has been used in many machine learning and natural language processing tasks. We can compute how close words represented by a word embedding are to each other, by calculating the cosine similarity (direction) of these vectors. A cosine similarity of 1 means it is the same word. A cosine similarity going towards 0 means the words are completely different.

However plagiarisms and similarity are more about sentences and paragraphs rather than just words. Therefore, we rather calculate the mean vectors of sentences and compare these vectors. During testing, we noticed that a cosine similarity  $> [0.85-0.90]$  is a rather reliable indication that sentences are very much alike.

Continuing on this essence we built on the idea and tokenized the input into sentences, each for which the mean vectors were computed. Then, each pairwise combination was checked for similarity and in case it matches over a certain threshold, then it was considered to account as a factor for the similarity score. In this way, with the help of weights assigned to each sentence on the basis of its length, the overall similarity score was calculated.

## 6.2.7 Jaccard Trigram

The approach and the calculations are the same as in the case of Cosine Trigram. We first calculate term frequency, document frequency, and inverse document frequency for both documents. Then weighted vectors are computed using values of term frequency and document frequency for both documents. Then Jaccard similarity is calculated between two weighted TF-IDF vectors. Jaccard score is calculated by taking the intersection of the two vectors and then dividing it by the union of these two vectors.

## 6.2.8 Longest Common Subsequence

It refers to the longest string of words that are the same between two documents. We then normalize this value by dividing the value by the total number of words in the answer [to be checked] document. We use the Dynamic Programming approach to find the value of LCS for two documents. This feature is very handy to account for actual cases of plagiarism.

## 6.2.9 N-gram Containment

An *n-gram* is a sequential word grouping. For example, in a line like "Bayes rule gives us a way to combine prior knowledge with new information," a 1-gram is just one word, like "Bayes." A 2-gram might be "Bayes rule" and a 3-gram might be "combine prior knowledge." Containment is defined as the intersection of the *n-gram* word count of a Source Text (S) with the *n-gram* word count of the Student Answer Text (A) *divided* by the *n-gram* word count of the Student Answer Text. If the two texts have no *n-grams* in common, the containment will be 0, but if *all* their *n-grams* intersect then the containment will be 1.

Here, different values of *n* help us in analysing the context of the inputs with respect to each other. For example, for smaller values of *n*, high containment value would indicate that the semantics are same for both the inputs and the inputs are indicating towards the same context, however for higher values of *n*, if the containment value is high, then there would be a direct case of plagiarism.

We have experimented with different datasets over different values of *n*, and understood the use cases, to use the values of *n* that are really helping us to gain a deeper understanding of different types of input.

## 6.2.10 NLTK Phrase Toolkit

This feature<sup>[8]</sup> is a combination of two factors : semantic similarity and word order similarity. Semantic similarity factor between two documents is computed by calculating the cosine similarity between semantic vectors of both the documents. These semantic vectors are formed by associating the value of each word in a vector that is of same size for the inputs, the size is equal to the union of words present in both the given inputs. In case the word from the joint set is not present in the input, then the word from the input which has the highest semantic similarity is associated as the weight of that

vector. This data of words and similarity is obtained from WordNet provided by NLTK [Natural Language Toolkit].

For the word order similarity, the weight of word for one vector is featured by its position and in case the word is not present, then the most similar word is given importance.

Both these factors are then given weightage on a linear scale to compute for the overall score. We experimented on what this weight should be and are using different weights for different use cases.

### 6.2.11 Rabin Karp

First, we filter documents by tokenizing and removing the stopwords. Then we calculate the hash value of each document and add it to the document type hash table. Hash Value is calculated using the concept of rolling hash. In rolling hash, we set the value of the rolling window for which hash values are calculated and keep adding it to the hash table. In our case, we set the value of the rolling window to be 3. To calculate the hash value for the next window we remove the value of the first word in the previous window and add the value of the last word in the current window. The formula for the rolling hash is designed such that only similar phrases have the same values. Then we calculate the plagiarism rate using values of hash for both the documents.

Formula for plagiarism rate =  $(sh / th\_a) * 100$

Where

sh = intersecting hash values between two documents

th\_a = number of hash values of document a.

### 6.2.12 Sequence Matcher

Sequence matcher is a tool available in the Python library 'difflib'. It is used to compute the longest matching character string and then give the computed score after normalization.

For the following input :


```
my stackoverflow mysteries
```

```
My.st.....er.....y.....
```

The output would be 'Mystery'

### 6.2.13 Tensorflow Universal Sentence Encoder

The Universal Sentence Encoder<sup>[9]</sup> encodes text into high dimensional vectors that can be used for text classification, semantic similarity, clustering and other natural language tasks. The model is trained and optimized for



greater-than-word length text, such as sentences, phrases or short paragraphs. It is trained on a variety of data sources and a variety of tasks with the aim of dynamically accommodating a wide variety of natural language understanding tasks. The input is variable length English text and the output is a 512 dimensional vector.

There are two Universal Sentence Encoders to choose from with different encoder architectures to achieve distinct design goals, one based on the transformer architecture targets high accuracy at the cost of greater model complexity and resource consumption. The other targets efficient inference with slightly reduced accuracy by the deep averaging network (DAN). We are using the model trained with a deep averaging network (DAN) encoder as we have already used BERT and also, to reduce the resource consumption. The input embeddings for words and bi-grams are first averaged together and then passed through a feedforward deep neural network (DNN) to produce sentence embeddings.

Both models were trained with the Stanford Natural Language Inference (SNLI) corpus. The SNLI corpus is a collection of 570k human-written English sentence pairs manually labeled for balanced classification with the labels entailment, contradiction, and neutral, supporting the task of natural language inference (NLI), also known as recognizing textual entailment (RTE). Essentially, the models were trained to learn the semantic similarity between the sentence pairs.

This encoder differs from the word level embedding models as it focuses on meaning of word sequences rather than just individual words.

## 6.3 Checking Plagiarism for documents

### 6.3.1 Real Plagiarism

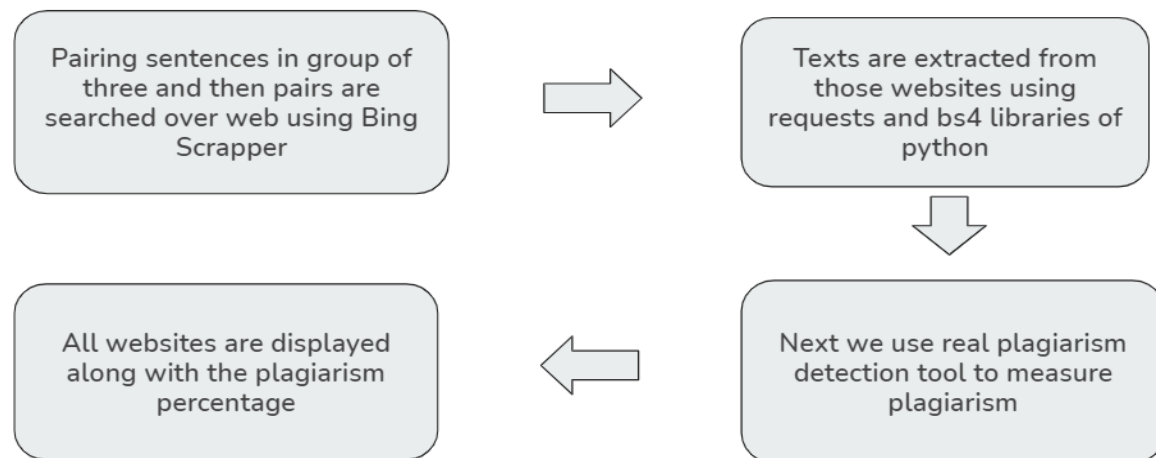
We worked on this aspect because of its high importance in the real world. By real plagiarism, we are referring to the plagiarism that should be penalised, it refers to the direct copying / no original thoughts scenarios. For this, we had to carefully consider the features that can contribute to such a measure. After experimenting over various datasets, we came to the conclusion that the following features with the weights as mentioned give an accurate measure of such plagiarism.

Feature	Weightage [in Percentage]
Cosine[2]	5
2-gram	15
3-gram	25
4-gram	20
Docsim	5
LCS	10
Sequence Matcher	20

The results for real plagiarism on Sheffield dataset

Extent of Plagiarism	Plagiarism Percentage
Cut and Paste	100
Light Revision	88.23
Heavy Revision	41.65
Non-Plagiarism	6.73

### 6.3.2 Online Plagiarism



This is a very important aspect of plagiarism as well. Students sometimes may take help from the Internet to cover for their assignments, tutorials and projects. We have followed a very systematic approach to resolve the same and to best identify all the cases of such plagiarism.

Given an input script, first we remove the stop words from the entire script. Then, the text is further processed by tokenizing into sentences. In case, the number of sentences are above a certain threshold [40], then we group the sentences into combinations of three to reduce the overall resources required to account for the entire text. Then, these chunks of text are searched over the web using Bing Scraper and all the top results for each chunk are then stored in a list.

These web searches are basically nothing but the websites which have the same context as those chunks of text. The next step is to extract the text from all these websites using `requests` and `bs4` libraries of Python.

Once, we have the text body of these relevant websites, then we can use the real plagiarism detection tool that we have built to check the plagiarism of this entire text with the text bodies obtained from these different websites.

Thus, we have been able to check the plagiarism of the given input across the web[Internet] by first identifying the possible sources and then checking each one, one by one systematically.

## Results :

- ❖ We obtained 100 percent plagiarism with Wikipedia in an article of Dr. APJ Abdul Kalam picked up from wikipedia

- ❖ Results for plagiarism with various websites for light revision data of the University of Sheffield are shown below. As it can be seen, we have got 15 percent plagiarism from wikipedia article, 82 percent plagiarism from a Gabormelli article and so on.

Text Field

The new classes, known as derived classes, take over (or inherit) attribute and behaviour of the pre-e

Predict

Reset prediction

**Online Plagiarism : [http://www.gabormelli.com/RKB/is-a\\_relation](http://www.gabormelli.com/RKB/is-a_relation) : [82.13985451173164]**  
**[https://en.wikipedia.org/wiki/Code\\_reuse](https://en.wikipedia.org/wiki/Code_reuse) : [15.863269165871706]**  
**[https://en.wikipedia.org/wiki/Inheritance\\_\(object-oriented\\_programming\)](https://en.wikipedia.org/wiki/Inheritance_(object-oriented_programming)) : [24.733563802631704]**  
**[https://www.academia.edu/271914/Object\\_Oriented\\_Software\\_Composition](https://www.academia.edu/271914/Object_Oriented_Software_Composition) : [2.053161810139869]**

### 6.3.3 Data Processing [for Further Work]

Owing to the fact that we have used multiple datasets, how we have used the data to modify to a certain format for the ease of model training and testing is a very important aspect of the project. We have used csv format to process the data to process the data systematically and have used Pandas library to make the necessary changes. At the end, there is a single .csv file which has names of all the input text and output text, with their respective file names linked with the class data [similarity scale], datatype label and the same source name and answer file name to avoid any unnecessary confusion.

Such format has been maintained by us in case of large uniform datasets such as STS benchmark dataset, SICK dataset and MSRRC dataset.

Further, to work on this dataset, it can easily be done by working on one row at a time and using several in-built functions in CSV and Pandas libraries.



	A	B	C	
1	File	Class	Datatype	Text
2	source1.txt		-1 orig	An air plane is taking off.
3	answer1.txt		5 train	A plane is taking off.
4	source2.txt		-1 orig	A man is playing a flute.
5	answer2.txt		3.8 train	A man is playing a large flute.
6	source3.txt		-1 orig	A man is spreading shredded cheese on an uncooked pizza.
7	answer3.txt		3.8 train	A man is spreading shredded cheese on a pizza.
8	source4.txt		-1 orig	Two men are playing chess.
9	answer4.txt		2.6 train	Three men are playing chess.
10	source5.txt		-1 orig	A man seated is playing the cello.
11	answer5.txt		4.25 train	A man is playing the cello.
12	source6.txt		-1 orig	Two men are fighting.
13	answer6.txt		4.25 train	Some men are fighting.
14	source7.txt		-1 orig	A man is skating.
15	answer7.txt		0.5 train	A man is smoking.
16	source8.txt		-1 orig	The man is playing the guitar.
17	answer8.txt		1.6 train	The man is playing the piano.
18	source9.txt		-1 orig	A woman is playing an acoustic guitar and singing.
19	answer9.txt		2.2 train	A man is playing on a guitar and singing.
20	source10.txt		-1 orig	A person throws a cat on the ceiling.
21	answer10.txt		5 train	A person is throwing a cat on to the ceiling.
22	source11.txt		-1 orig	The man spanked the other man with a stick.
23	answer11.txt		4.2 train	The man hit the other man with a stick.
24	source12.txt		-1 orig	A woman picks up and holds a baby kangaroo in her arms.
25	answer12.txt		4.6 train	A woman picks up and holds a baby kangaroo.
26	source13.txt		-1 orig	A man is playing a bamboo flute.
27	answer13.txt		3.867 train	A man is playing a flute.
28	source14.txt		-1 orig	Someone is folding a piece of paper.
29	answer14.txt		4.667 train	A person is folding a piece of paper.
30	source15.txt		-1 orig	A panda dog is running on the road.
31	answer15.txt		1.667 train	A man is running on the road.
32	source16.txt		-1 orig	A dog is trying to eat the bacon on its back.
33	answer16.txt		3.75 train	A dog is trying to get bacon off his back.
34	source17.txt		-1 orig	A polar bear is sliding across the snow.
35	answer17.txt		5 train	The polar bear is sliding on the snow.
36	source18.txt		-1 orig	A woman is swimming.
37	answer18.txt		0.5 train	A woman is writing.

< file\_information

Sheet 1 of 1 Default English (Indi

## 6.3.4 Semantic Plagiarism

We tried two different models

### 6.3.4.1 Binary Classification

#### 6.3.4.1.1 Dataset1

We used two datasets for training and testing our model. We used the MSRPC dataset<sup>[1]</sup> and the University of Sheffield dataset<sup>[4]</sup>. Every text file is compared with the source file and is classified whether it is plagiarized or not. The given data is divided into test and training datasets and further processing is done. We also used this model on our AI dataset.

### 6.3.4.1.2 Model

We are using a binary classification model designed using PyTorch. The input to the model is the similarity features that we are using to measure similarity among documents and output is a single sigmoid value that is rounded to the label 0 or 1, classifying the plagiarism. 1 here indicates that the document is plagiarized while 0 indicates it is not plagiarized. We tried the different combinations of nodes in hidden layers and got results respectively. Results are shown in the table below for different trials in the model. Dropout was set at 0.2. The learning rate was set at 0.001.


**Rectified Linear Unit(ReLU) is being used as the activation function.** It is a piecewise linear function that will output the input directly if positive else it will output zero.

**Adam<sup>[10]</sup> was used as the optimizer.** It is an optimization algorithm and can be used instead of stochastic gradient descent to update network weights on basis of training data. The name Adam<sup>[10]</sup> is derived from adaptive moment association. Adam<sup>[10]</sup> is different from classical stochastic gradient descent(SGD)<sup>[12]</sup>. In the case of stochastic gradient descent, it maintains a single learning rate for all weight updates and maintains the same learning rate throughout the training. While in the case of Adam<sup>[10]</sup> learning rate is maintained for each network weight and is adapted separately as the model learns. Adam<sup>[10]</sup> can be defined as combining the advantages of two other extensions of stochastic gradient descent which are:

**Adaptive Gradient Algorithm:** this maintains a per-parameter learning rate and improves performance on problems with sparse gradients, for example, natural language and computer vision.

**Root Mean Square Propagation:** this also maintains a per-parameter learning rate that adapts on basis of the average of recent magnitudes of the gradients for the weight

Instead of adapting the parameter learning rates on the basis of the average first moment (the mean) as in the case of RMSProp, Adam<sup>[10]</sup> also makes use of the average of the second moments of the gradients (the uncentered variance).



The algorithm calculates an exponential moving average of the gradient and the squared gradient, and the parameters beta1 and beta2 control the decay rates of these moving averages. The initial value of the moving averages and beta1 and beta2 values close to 1.0 (recommended) result in a bias of moment estimates towards zero. This bias is overcome by first calculating the biased estimates before then calculating bias-corrected estimates.

**Binary Cross-Entropy loss (BCELoss) was used as the loss function.** It is the default function to use in the case of binary classification. The Cross-entropy will calculate a score that summarizes the average difference between the actual and predicted probability distributions for predicting class 1. The score is minimized and a perfect cross-entropy value is 0.

## 6.3.4.2 Documents Similarity


### 6.3.4.2.1 Data

Every text file is compared with the source file and is given a score on a scale of 1 to 5 for the similarity between the two files. We trained and tested our model on the SICK dataset<sup>[2]</sup> and obtained pretty good results. The given data is divided into test and training datasets in a ratio of 20:80 and further processing is done.

### 6.3.4.2.2 Model

We are using a regression model designed using PyTorch. The input to the model is the similarity features that we are using to measure similarity among documents and output is a single value on a scale of 1 to 5 indicating the similarity between documents. We tried different numbers of nodes in hidden layers and got results respectively. Results are shown in the table below for different trials in the model. The learning rate was set at 0.005.

**Sigmoid is being used as the activation function.** The input to the function is transformed into a value between 0.0 and 1.0. Inputs that are much larger than 1.0 are transformed to the value 1.0, similarly, values much smaller than 0.0 are snapped to 0.0



**Stochastic Gradient Descent was used as the optimizer.** Gradient Descent<sup>[12]</sup> is an algorithm that is used to find the value of parameters of a function to minimize the cost function. The aim of the algorithm is to try different values for coefficients and evaluate the cost and select those coefficients which lower the value of cost. After certain repetitions, we will get the best value of coefficients for which the value of cost will be the lowest. The process starts by assigning a small random value to coefficients and then the value of cost is calculated by substituting the value of the coefficient. Then derivative of the cost function is taken. We need to know the slope so that we can move coefficient values in a certain direction so that we get lower costs in the next iteration. The process is repeated until the value of the cost function is close to zero. The learning rate is also specified to control the change in coefficient values in each update.

Gradient Descent<sup>[12]</sup> can take a lot of time to run on very large datasets. Because gradient descent algorithm requires prediction for every instance of training dataset and there can be millions of instances. In the case of a large amount of data, we can use stochastic gradient Descent<sup>[12]</sup> a variation of gradient descent. In this case, the update is made in the value of coefficient after each training instance rather than after each batch. The first step is to randomize the order of the training dataset. This is to mix the order in which the updates are made to coefficients. As the coefficients are updated after every training instance the updates will be noisy and jumping. So by mixing the order of updates to the coefficients it harnesses this random walk and avoids it from getting distracted or stuck. The learning is much faster with stochastic gradient descent for a large training dataset.

**MSE (MSELoss) was used as loss function.** MSELoss is mainly used for regression problems. Mean squared error is computed by taking the average of the squared differences between the predicted and actual value. The obtained result is always positive irrespective of the sign of predicted and actual value. The squaring indicates that larger mistakes result in more error than smaller mistakes and thus model gets punished more for larger mistakes.

## 6.3.5 Text Entailment

### 6.3.4.1 Data

Every text file is compared with the source file and is classified in to three different classes “Contradiction”, ”Entailment”, and “Neutral”, depending on two files. We trained and tested our model on the SICK<sup>[2]</sup> dataset and obtained pretty good results. The given data is divided into test and training datasets and further processing is done.

### 6.3.4.2 Model

We are using the Multiclass Classification Model<sup>[15]</sup> defined using Keras. The input to the model is the similarity features that we are using to measure similarity among documents and output is a single class of the three classes “Contradiction”, ”Entailment”, ”Neutral” indicating the text entailment between documents. We tried different numbers of nodes in hidden layers and got results respectively. Results are shown in the table below for different trials in the model. The output layer has three nodes which is equal to number of classes.

**Rectified Linear Unit(ReLU) is being used as the activation function in the hidden layer.** It is a piecewise linear function that will output the input directly if positive else it will output zero.

**Softmax is being used as the activation function in the output layer.** The softmax<sup>[10]</sup> function is used as the activation function in the output layer of neural network models that predict a multinomial probability distribution. That is, softmax is used as the activation function for multi-class classification problems where class membership is required on more than two class labels. The softmax activation will output one value for each node in the output layer. The output values will represent probabilities and the values sum to 1.0.

**Adam<sup>[13]</sup> was used as the optimizer.**

**Categorical Cross Entropy was used as the loss function.** It is the default loss function to use in the case of the multiclass classification problem. Categorical Cross-entropy will calculate a score that summarizes the average difference

between the actual and predicted probability distributions for all classes in the problem. The score is minimized and a perfect cross-entropy value is 0. This function requires that the output layer is configured with n nodes, one for each class and a 'softmax' activation function to predict the probability of each class.

### Entailment Analysis:

**Text 1 :** A man with a jersey is dunking the ball at a basketball game

**Text 2 :** The ball is being dunked by a man with a jersey at a basketball game

Entailment Analysis Result : **Entailment**

**Text 3 :** Two children are lying in the snow and are making snow angels

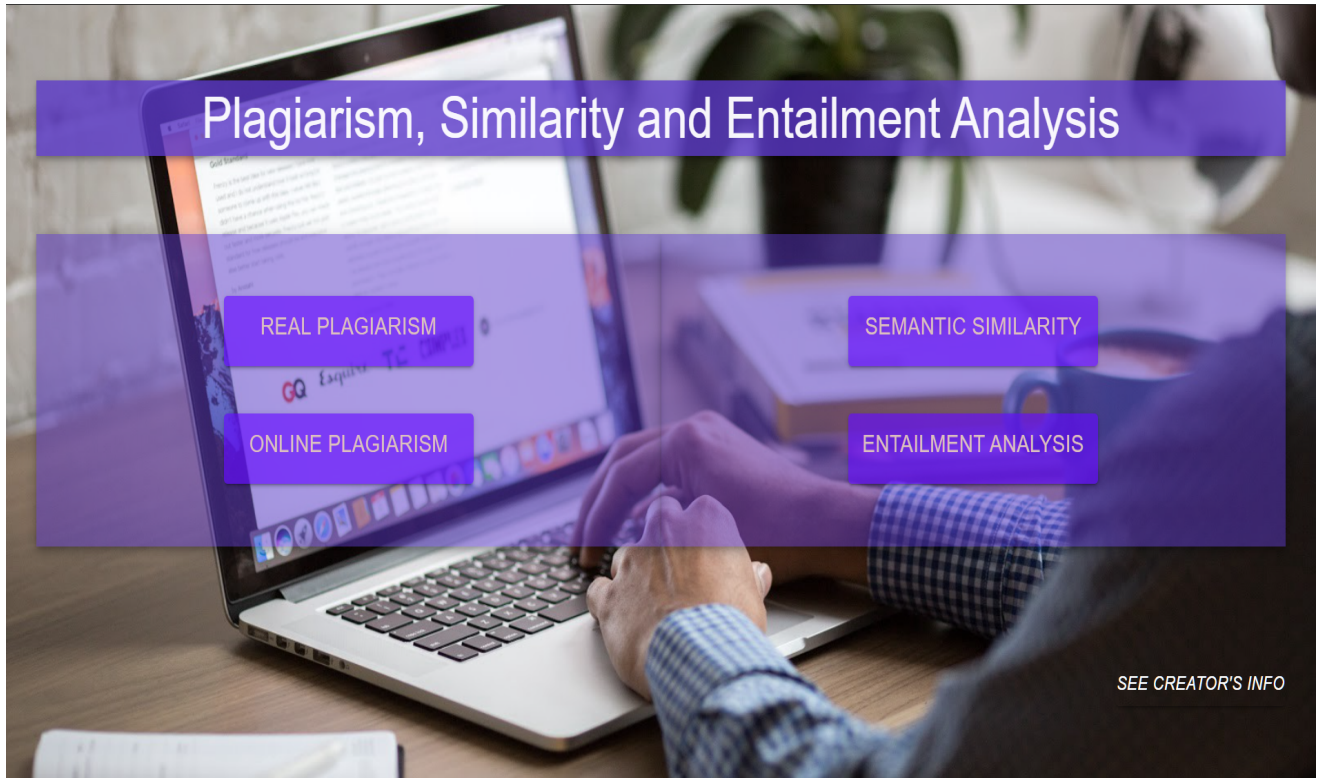
**Text 4 :** There is no child lying in the snow and making snow angels

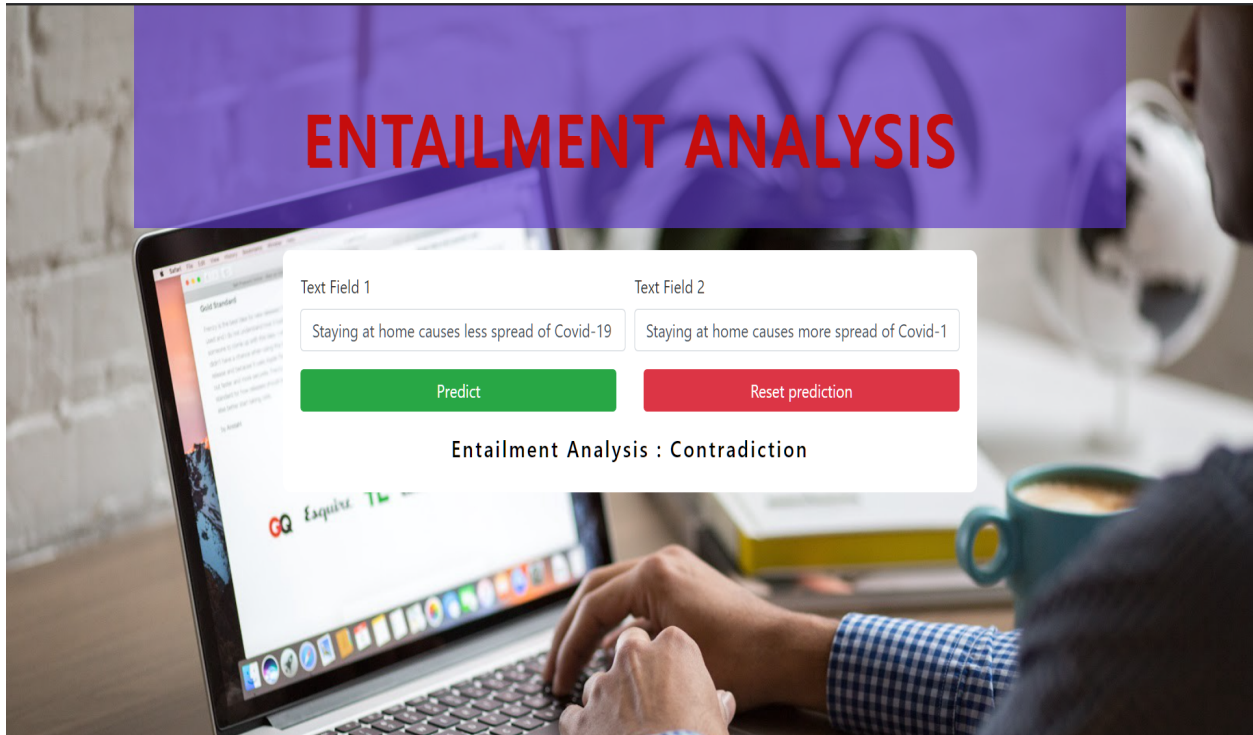
Entailment Analysis Result : **Contradiction**

## 7 Website Integration

- ❖ We built a compact and user friendly website for easy testing and for convenience of users.
- ❖ We have used HTML, CSS, and React Framework for the Front End part
- ❖ We have used Flask for the backend part
- ❖ We made a request to Flask instance to get the results by running the Python script in the background and then getting the results and displaying them on the screen.







## 8 User Guidelines

The user must submit all the documents in the respective folders according to the formats. Then, the documents should be converted in a single format after running the respective codes required to change the formats.

Then run the python scripts for online plagiarism detection, in between plagiarism detection, and semantic similarity analysis, and textual entailment analysis on the documents as needed.



## 9 Future Prospects

- ❖ Clustering of documents to reduce the total number of pairs required to be checked for similarity/plagiarism detection by large numbers
- ❖ Embedding a knowledge graph for enhanced idea-plagiarism checking
- ❖ Automatic Grading of answer scripts
- ❖ More features integrated website and mobile application

## 10 Result

- ❖ The binary classification model achieved an accuracy of 78.2%
- ❖ The Mean Square Error for the regression model was 0.315
- ❖ The multiclass classification model achieved an accuracy of 82.4%

## 11 Techstack

- ❖ Programming languages : Python
- ❖ DL libraries : Tensorflow, Keras, Pytorch
- ❖ Text Correction libraries: TextBlob, Pyspellchecker, Symspell
- ❖ Text Extraction : Microsoft Computer Vision API
- ❖ PDF to text Conversion Work : pdfplumber, pdf2image
- ❖ Frontend : HTML, CSS , ReactJS
- ❖ Backend : Flask

## 12 Demo and Github Link

Demo Video Link :

[https://drive.google.com/drive/folders/1MAr1qtvh\\_NnigaQ2oUsfLvLEE7u\\_ad19?usp=sharing](https://drive.google.com/drive/folders/1MAr1qtvh_NnigaQ2oUsfLvLEE7u_ad19?usp=sharing)

Github Codebase Link :

[https://github.com/vyaskartik20/Semantic\\_Plagiarism\\_Checker\\_for\\_Handwritten\\_Scripts](https://github.com/vyaskartik20/Semantic_Plagiarism_Checker_for_Handwritten_Scripts)

## 13 Conclusion

We were successfully able to extract text from scanned handwritten documents using Microsoft Computer Vision API<sup>[1]</sup> and further improved obtained text using Textblob, Pyspellchecker, and Symspell<sup>[7]</sup> library. Next, we developed different similarity features to calculate the similarity between documents. We designed three different models, binary classification model, regression model, and multiclass classification model. We trained these models on MSRPC<sup>[3]</sup>, SICK<sup>[4]</sup>, and STS<sup>[6]</sup> datasets and obtained great results. We were able to predict similarity on a binary basis using the binary classification model also we were able to predict similarity on a scale of 1-5 using the regression model.

We were also able to predict text entailment using a multiclass classification model. We also developed a real plagiarism detection tool that indicates actual copying between documents. We also developed a tool to detect pages on the web from where documents were plagiarized. In the end, we also developed a website where all these features can be used in one place and similarity/plagiarism can be calculated between documents.

## 14 Acknowledgments

We are heartily thankful to our instructors, Dr. Romi Banerjee and Dr. Debarati Bhunia Chakraborty, for providing us the necessary guidance, the needed constant support, and helping us throughout the course of the project via continuous interaction and evaluation of the course at regular intervals.

## 15 References

- 1) [MSRPC dataset](#)
- 2) [SICK dataset](#)
- 3) [STS dataset](#)
- 4) [University of Sheffield Dataset](#)
- 5) [Microsoft Computer Vision API](#)
- 6) [Symspell](#)
- 7) [BERT sentence Encoder](#)
- 8) [NLTK Toolkit](#)
- 9) [Universal Sentence Encoder | TensorFlow Hub](#)
- 10) [Adam](#)
- 11) [Regression](#)
- 12) [Stochastic Gradient Descent](#)
- 13) [How to choose loss functions](#)
- 14) [Developing-a-corpus-of-plagiarised-short-answers.pdf](#)
- 15) [Multi-Class Classification](#)
- 16) [How to Develop Deep Learning Models](#)
- 17) [Softmax Activation Function.](#)
- 18) [Academic Plagiarism Detection](#)