COMPUTOLOGY: Journal of Applied Computer Science and Intelligent Technologies
Volume 4, Issue 2, Jul-Dec 2024, pp. 61-88
DOI: 10.17492/computology.v4i2.2404
https://www.journalpressindia.com/cjacsit

Plagiarism Detection using Machine Learning Techniques and Cosine, Jaccard and Dice Similarity Measures

Shahzeb Khan, Deepankar Krishna**,*
*Samrailatpam Mukherjee**, Rohit Kumar** and Mohd Tajammul***

ABSTRACT

With abundant textual content available on the internet, Plagiarism detection has become very important to safeguard original works, and reduce plagiarized content on the internet. In this research we present potential methods of plagiarism detection for textual documents using a corpus that mimics different levels of plagiarism committed by students in academics, assignments and theses by using Machine Learning Classifiers based on similarity features - cosine, jaccard similarity and dice coefficient values of textual documents. We used 2 strategies (models), the first strategy only uses jaccard and dice coefficient for training and testing which resulted in an accuracy score of 78.95%. The second strategy was created as a consequential improvement of the first strategy by involving all three i.e cosine, jaccard and dice coefficient for both target feature creation and for training and testing. Our research reveals that the Strategy-2, proves to be an improvement of Strategy-1, achieving an accuracy score of 97.31%.

Keywords: *Plagiarism; Vectorization; TF-IDF; Word embedding; Cosine similarity; Jaccard similarity; Dice coefficient.*

1.0 Introduction

In our modern digital world which is built on information, and with the ever increasing

**Corresponding author; Assistant Professor, Department of Computer Science and Applications, Sharda University, Greater Noida, Uttar Pradesh, India (E-mail: shahzeb.khan@sharda.ac.in)*
***Student, Department of Computer Science and Applications, Sharda University, Greater Noida, Uttar Pradesh, India (E-mail: 2023564421.deepankar@ug.sharda.ac.in; 2023420324.samrailatpam@ug.sharda.ac.in; 2023547376.rohit@ug.sharda.ac.in; mohd.tajammul@sharda.ac.in)*

amount of material available on the internet, it has become extremely important to Identify whether a particular text is plagiarized or not, in order to protect the content creators. For this reason, plagiarism detection systems are essential. Plagiarism Detection (PD) can be defined as the procedure that finds similarities between a document and other documents based on lexical, semantic, and syntactic textual features (Ali & Taqa, 2022).

This research proposes a plagiarism detection model implemented in Python programming language, designed to evaluate the originality by giving the level of plagiarism contained in a textual document by utilizing a combination of NLP (natural language processing), preprocessing techniques, vectorization using vector space models, and multiple similarity measures. Our approach involves cleaning and standardizing text through preprocessing steps, followed by transforming the text data into numerical representations using the Term Frequency-Inverse Document Frequency (TF-IDF) as our vector space model along with similarity measurement techniques - cosine similarity, jaccard similarity and dice coefficient, we extract numerical features of our textual data. Vector space model (VSM), helps us to convert the original string text within a document into a vector of numbers. (Afzali & Kumar, 2017)

To assess textual similarity, we use *cosine similarity, Jaccard similarity, and Dice coefficient*, comparing the original files to the copied files and deriving numerical similarity with all these *similarity measures*.

Cosine Similarity: Cosine similarity measures the cosine of the angle between the vectors in a multidimensional space (Sari, 2023). This similarity measure represents text documents as vectors and calculates how similar these vectors are in a multidimensional space by representing these vectors as points, and calculating the cosine angle between these vectors. The text data is transformed into numerical vectors using vectorization techniques such as TF-IDF (Term Frequency-Inverse Document Frequency).

$$Cosine\ Similarity = \frac{A \cdot B}{||A||\ ||B||} \quad \ldots 1$$

Where A and B are 2 vectors (textual documents)

A·B is the dot product of the two vectors. And $||A||\ ||B||$ is the product of the magnitudes (lengths) of A and B. If the angle between two vectors is small (cosine similarity close to 1), then they would be more similar. If the angle between the two vectors is large (cosine similarity close to 0), then they would be less similar. The cosine similarity value ranges between -1 to 1, where a value of -1 means the text is non similar and value of 1 means totally identical.

Jaccard Similarity: The jaccard similarity measures the ratio of the values common in two sets (intersection) and the combined values of both sets (union).

It is like finding out which fruits are common in two fruit baskets with many different fruits and then finding out how many combined fruits are there in both baskets, the ratio will give us the measure of to what extent there are common fruits in both baskets. If A and B are the vectors of two textual documents then the jaccard similarity between A and B is calculated as given below:

$$Jaccard\ Similarity = \frac{|A \cap B|}{|A \cup B|} \quad \ldots 2$$

The range of values in Jaccard Similarity is between 0 to 1, with a value of 0 or near 0 being non similar, and value 1 or near 1 being identical texts.

Dice Coefficient: Dice coefficient is similar to jaccard similarity measure but it gives more importance to the intersection of the two sets, and does not use union between the sets but the total values in the set. Dice coefficient compares the size of the intersection to the total size of both sets combined.If A and B are the vectors of two textual documents then the Dice coefficient between A and B is calculated as given below: **(Equation 3):**

$$Dice\ Coefficient = \frac{2\,|A \cap B|}{|A| + |B|} \quad \ldots 3$$

|A| and |B| are the size of the two sets A and B, so if A = {1,2,3} and B = {2,3,4}, both of them have 3 elements each so |A| + |B| = 3 + 3 = 6.

The Dice coefficient value map ranges from [0, 1], where 0 represents non-overlapping and 1 represents a perfect agreement (Afzali & Kumar, 2017).

NOTE: Although the cosine similarity value ranges from -1 to 1, to keep the scales same for all three similarity measures the cosine similarity value range has been re-scaled between 0 to 1 instead of -1 to 1. This is done as the jaccard and dice coefficient also have 0 to 1 as the ranges. This means that now if cosine similarity value gives 0, it means the texts are non similar, and 1 means the texts are identical.

This paper introduces a new plagiarism detection model that will help in detecting the level of plagiarism in a textual document which will help in protecting content creators and original academic works.

NOVELTY: Our work initially began with the aim of only using cosine similarity as a similarity measure when we were required to set up target feature - which is the level of plagiarism[strategy 1] to set that up and use jaccard similarity and Dice coefficient as features for training and testing. Which gave us a maximum accuracy score of 78.95%. Then we thought about designing the target feature where all three similarity measures have equal say in determining the target feature. So then we created strategy 2 as a consequential improvement of strategy one, in which all three were used both for setting up the target feature (level of plagiarism) and then training and testing of the model uses not just jaccard similarity and dice coefficient but Cosine Similarity as well. And the

accuracy scores achieved were 97.31%. Making a significant improvement. This demonstrates the potential of our novel strategy to significantly improve text analysis and plagiarism detection systems by leveraging the strengths of diverse similarity measures and inclusion of cosine similarity as measure in creation of textual plagiarism detection in academic works.

2.0 Related Work

There have been many different types of plagiarism detection models proposed, from text to source code to images, in our work, we are primarily focusing on textual plagiarism detection and work (Ali & Taqa, 2022) discusses different types of plagiarism, and the approaches for numerical representation of textual documents which is used for making the textual documents comparable using similarity measures such as cosine and jaccard. In work (Sadhin *et al.*, 2024) some algorithms like SVM, Xgboost, Naive Bayes, and Logistic regression are used, on the attributes derived from programming assignments where SVM and Xgboost perform significantly better. Plagiarism can be of many types, one of which is paraphrasing about which work (Kholodna *et al.*, 2022) develops a machine learning model based on Siamese Neural Network.

How machine learning algorithms are implemented using text based methods such as TF-IDF vectorization and cosine similarity and neural networks are described in the work (Husain *et al.*, 2024) in which SVM and Random Forest Perform very well giving an accuracy of 92%. Work (Sari, 2023) demonstrates use of TF-IDF and cosine similarity for detecting plagiarism on academic thesis documents. Work (Yülüce & Dalkılıç, 2022) Also uses TF-IDF vectorizer along with SVM, Gaussian Naive Bayes, Multilayer perceptron and Linear Regression for author identification on a dataset of Turkish newspaper articles.

Work (Hunt *et al.*, 2019) creates a Plagiarism Detection model for Paraphrasing Identification using SVM, Linear regression. The work (Afzali & Kumar, 2017) uses similarity measures as ours i.e. Cosine Similarity, Jaccard Similarity and Dice Coefficient to understand which similarity measure works best among the three, among four textual datasets, revealing the most accurate results are given by cosine followed by jaccard and then Dice, this finding was crucial in development of strategy 1 in our research.

Work (Zechner *et al.*, 2009] elucidates about external and intrinsic plagiarism detection, where there are techniques described to speed up the process of external plagiarism detection, and for intrinsic plagiarism detection they try to find unusual writing styles within a text document. In work (Saeed, 2023) two plagiarism detection models are

proposed, the first one based on Exact, Lexical, Semantic, Merged Lexical- Semantic Plagiarism detection and the second model based on Same Weight Lexical, Same Weight Semantic, Variant Weight Lexical and Variant Weight Semantic Plagiarism Detection algorithms.

Work (Shahmirzadi *et al.,* 2019) evaluates the performance of various text vectorization methods such as simple TF-IDF to LSI topic model, D2V neural Model concluding that simple TF-IDF is a sensible choice for vectorization given its performance and cost. A plagiarism detection model based on paraphrasing has been proposed in work (Chitra & Rajkumar, 2016) using SVM, on lexical, syntactic and semantic features. Work (Alfikri & Purwarianti, 2014) proposes an extrinsic plagiarism detection model which uses features like word similarity, fingerprinting, and LSA (latent semantics analysis) using Naive Bayes and SVM achieving an accuracy of 95% on SVM. Although similarity measurement is a great tool for plagiarism detection, there have been questions raised about similarity measurement.

Work (Subroto & Selamat, 2014) argues that similarity measurement might not be enough for accurate plagiarism detection as similar sentences might still have different meaning and instead propose machine learning approaches as a solution by creating hybrid models using KNN, SVM and ANN (artificial Neural Network), concluding that the hybrid ANN-SVM is the method best performing for plagiarism detection.

Advanced language models like BERT and ROBERTa and GloVe are used in work (Rosu *et al.*, 2020), which, as proposed in work (Rosu *et al*., 2020), performs better than traditional word matching for plagiarism detection, concluding that BERT offers best results in terms of understanding the context.

Sometimes Code can be plagiarized and in work (Bandara & Wijayarathna, 2011) Naive Bayes Classifier, K Nearest Neighbor Algorithm and Ada boost algorithms are used to identify whether a piece of code written by students enrolled in programming courses is plagiarized or not based on features like lines of code, line length, access specifiers, frequency of comments, specifically in Java Programming language.

In work (Sabri *et al.*, 2022) Algorithms like SVM, Decision Tree, Random Forest, K-Nearest Neighbour (KNN) and logistic regression, are used along three feature vectorization methods such as word count, TF-IDF and word2vec, for text categorization in arabic language, belonging to different topics like ‘sports’, ‘business’, ‘Entertainment’, ‘Middle east’, ‘Scitech’, ‘history’ with logistic regression giving the highest score for all three vectorization methods.

Different words that have the same meaning also are used in the same context. The work (Rogozin *et al.*, 2019) Involves designing a convenient and fast system, for

finding similar text using word2vec, that learns how words appear together, for eg "Bread" and "butter", to understand the relationship between words, Then Doc2vec is used for learning how interrelated two documents are, to find similar documents, and achieve an accuracy of 90%. Recommender systems recommend objects for example news, movies, books, media, based upon the similarity measures. This similarity measure can be calculated as the numerical distance between multiple objects. In work (Pereira *et al.*, 2017) Cosine similarity is used as the measure of similarity between two vectors of n dimensions. The model uses past user activities and predicts items a user might be interested in.

Work (Tessari *et al.*, 2024) describes that For comparison of multidimensional quantities common measures like Cosine Similarity, Euclidean distance and Manhattan distances are used but their applicability reduces as the number of dimensions increases. The research gives limitations of cosine similarity and also suggests a new metric called Dimension Insensitive Euclidean Metric (DIEM) that could be more accurate and to analyze multidimensional data. Software plagiarism occurs when someone else's source code is copied and disguised as the original software, which can be done in different programming languages making it even harder to detect plagiarism. To address this issue work (Ullah *et al.*, 2021) uses PCA (principal component analysis) for extracting common features and then MLR (Multinomial Logistic Regression) classifies whether the code is plagiarized or not. In work (El-Rashidy *et al.*, 2022) a database consisting of all features that reflect different types of text similarities is created. Intelligent Deep learning techniques such as Convolutional and recurrent neural networks are applied for creation of a plagiarism detection model.

Work (Zahid *et al.*, 2023) Proposes a model to detect Cross Lingual Plagiarism, which happens when content is translated from one language to another. In the work, Urdu- English text dataset is used, and then cosine and jaccard similarity for features. Five machine learning algorithms are applied including KNN, Naive Bayes, SVM, Decision Tree, and Random Forest. Results reveal that Out of the five, KNN and Random Forest performed better.

In work (Sahu, 2016) plagiarism is defined as the act of taking someone else's work and passing it off as your own without giving credit to the original writer. Work (Sahu, 2016) Compares two files and calculates how many words are similar. KNN (K's nearest neighbor) is applied to classify a piece of work and compare it with the stored paper in the database to find out the plagiarism.

3.0 Methodology

3.1 Dataset description

1. Data collection source

The dataset that we had used in this research is a version created by Paul Clough and Mark Stevenson in the work (Clough & Stevenson, 2009), in which the creation of the dataset is thoroughly described.

2. Dataset overview

The corpus is designed to represent the types of plagiarism that are found within higher education as closely as possible which makes it suitable for training and evaluating a machine learning model which aimed at detecting plagiarism in textual documents specifically for academic texts.

3. Purpose

The dataset facilitates the development and assessment of algorithms for identifying instances of text similarity and potential plagiarism. The primary goal is to enable the model to differentiate between original content and copied or paraphrased text.

4. Content

Text Data: The dataset consists of a collection of two different types of documents files one type of files are the original and other types of files are a copied version of the original. The copied version of the original files have four levels plagiarism

a. NEAR COPY (CUT): these are the files that are nearly the same or identical as the original text files.
b. LIGHT REVISION: In this, the text has been altered in some basic ways like word substitutions, using synonyms and paraphrasing etc.
c. HEAVY REVISION: In the files that are heavy revision of the original texts, different words and different sentence structure is used, and is heavily rephrased but conveys the same meaning.
d. NON – PLAGiARIZED: The text files are entirely newly created and are not paraphrase or rephrased of the original texts.

5. Vectorization

Method: The text data is transformed into numerical vectors using vectorization techniques such as TF-IDF (Term Frequency-Inverse Document Frequency).

Output: The vectorized representation captures the semantic and syntactic features of the text, facilitating similarity comparison.

6. Dataset format

File format: The original format is of raw text (.txt files) but after preprocessing, the dataset may be provided in formats such as CSV, or text files, where each includes the text content and associated labels.

7. Structure of the dataset

There are 5 original text files 'taska', 'taskb', 'taskc', 'taskd', 'taske' - each of which contain some text written in plain english language regarding different random topics, the dataset is in raw text. And correspondingly there are 95 copied files, each of which have copied or paraphrased text from the original file. So all the original text files 'taska', 'taskb', 'taskc', 'taskd', 'taske' have 19 corresponding copied text files each, totalling to 95 copied text files. Also in raw text.

8. Data cleaning and preparation

Since the dataset is in raw text, it requires cleaning and preprocessing and then vectorization for further progress.

3.2 Preprocessing and NLP techniques

Preprocessing refers to the steps and techniques used to clean, transform, and organize raw data before it is fed into a machine learning model or analysis. Preprocessing is required to be performed on the dataset to convert the raw textual into data that can be used for analysis. In this research we are using a textual data set, which comprises 5 original text files and 95 copied/plagiarized text files.

Steps of preprocessing before we do any analysis will be as follows:

3.2.1 Conversion to lowercase

This involves converting all the text in lowercase letters. We do this to make texts across all files comparable, otherwise if an original text file has the sentence "APPLES ARE TASTY", and the copied file has the sentence "apples are tasty", then they might be treated as different sentences which is not the case.

3.2.2 Tokenization

Paragraphs are broken down into sentences, and sentences are broken into individual words, and each word is a 'token'. For example - the sentence "apples are tasty. mangoes are tastier" will be tokenized as - ["apples", "are", "tasty", "mangoes", "are", "tastier"]. Notice that even the punctuation is tokenized (period symbol (.)), which does not provide much idea about plagiarism in the content as all sentences have punctuations.

3.2.3 Punctuation removal

Since punctuation marks are not very important, we remove the punctuation marks from our tokenized dataset. All sorts of punctuations are removed to make the text more consistent and clean across all files, which will facilitate in more accurate comparison.

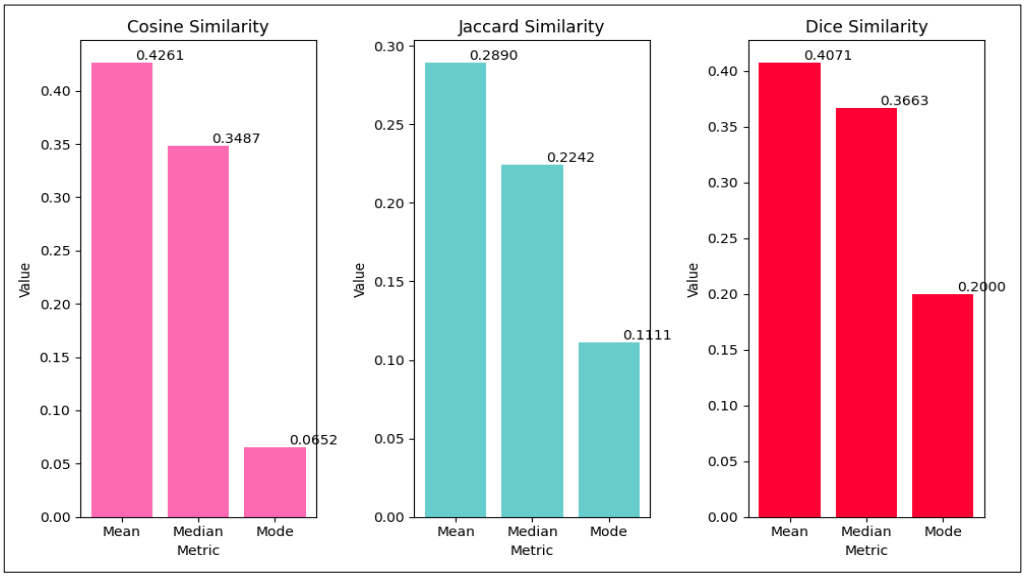
3.2.4 Stemming and lemmatization

Stemming is the process of removing the suffixes from words, For example - "happening" after stemming will become "happen", or "removing" will become "remov", but since "remov" is not a word, it will require further processing, which brings us to the next step Lemmatization. Lemmatization takes the word after the stemming process, for example "remov" (Stemming of the word "removing"), and then searches the closest matching word in the dictionary of the English language. Keep in mind that Lemmatization requires you to specify the Language for the dictionary, which in this case is English. After preprocessing steps described above our dataset is ready for analysis.

3.3 Statistical analysis

To perform statistical analysis **(Figure 1)** on textual data first the data has to be preprocessed. After preprocessing the data, Feature extraction needs to be performed. Meaning it has to be converted into numerical form through vectorization.

Figure 1: Statistical Analysis of Cosine, Jaccard and Dice similarity



Here we use the TF-IDF (Term Frequency-Inverse Document Frequency) vectorizer. After vectorization we calculate similarity values based on cosine similarity, jaccard coefficient and dice coefficient for each of the 95 copied files with respect to the 5 original text files. Then on these similarity values we perform statistical analysis and calculate mean, median, mode. As we can observe, most of the values are around 0.40 on a scale of 0.0 to 1.0 with 0.0 being non plagiarized, and 1 meaning Plagiarized, This means that a simple binary classification (<0.5 means Non-plagiarized & >0.5 means Plagiarized) will not work, since most of the data will come under the category of Non Plagiarized. This can lead to Overfitting issues in the dataset while training the model. This means that Classification of plagiarized text needs to have multiple fine grained categories to train the model properly.

3.4 Graphical analysis

We created heat maps based on Cosine similarity, for each task- a,b,c,d,e. In the given heatmap the cosine similarity of orig_taska.csv to orig_taska.csv is not given as it would be unnecessary, as comparison of a file with itself will always result in a score of 1.00.

Figure 2: Document Similarity Heatmap of Taska

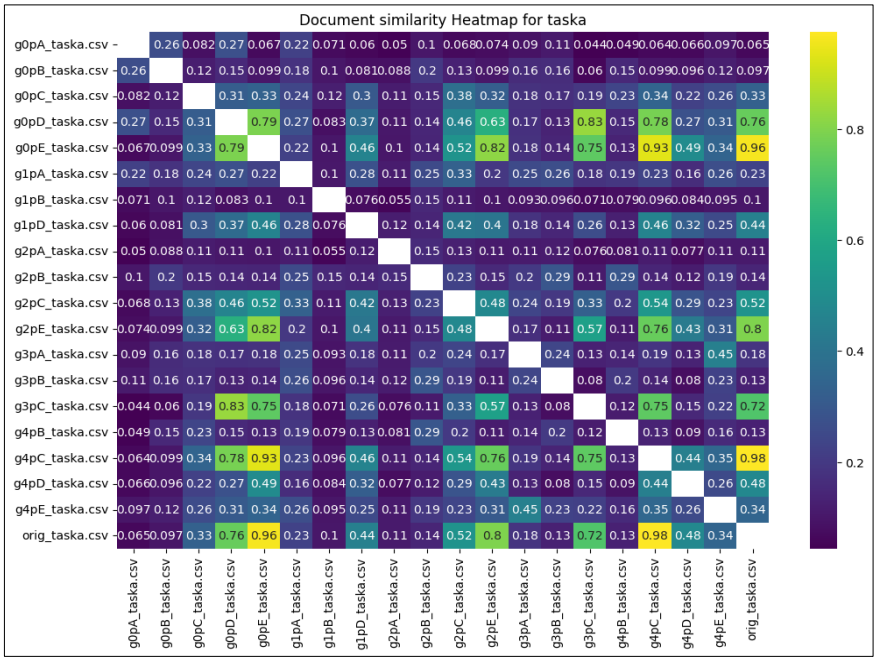


Figure 3: Jaccard Coefficient Heatmaps

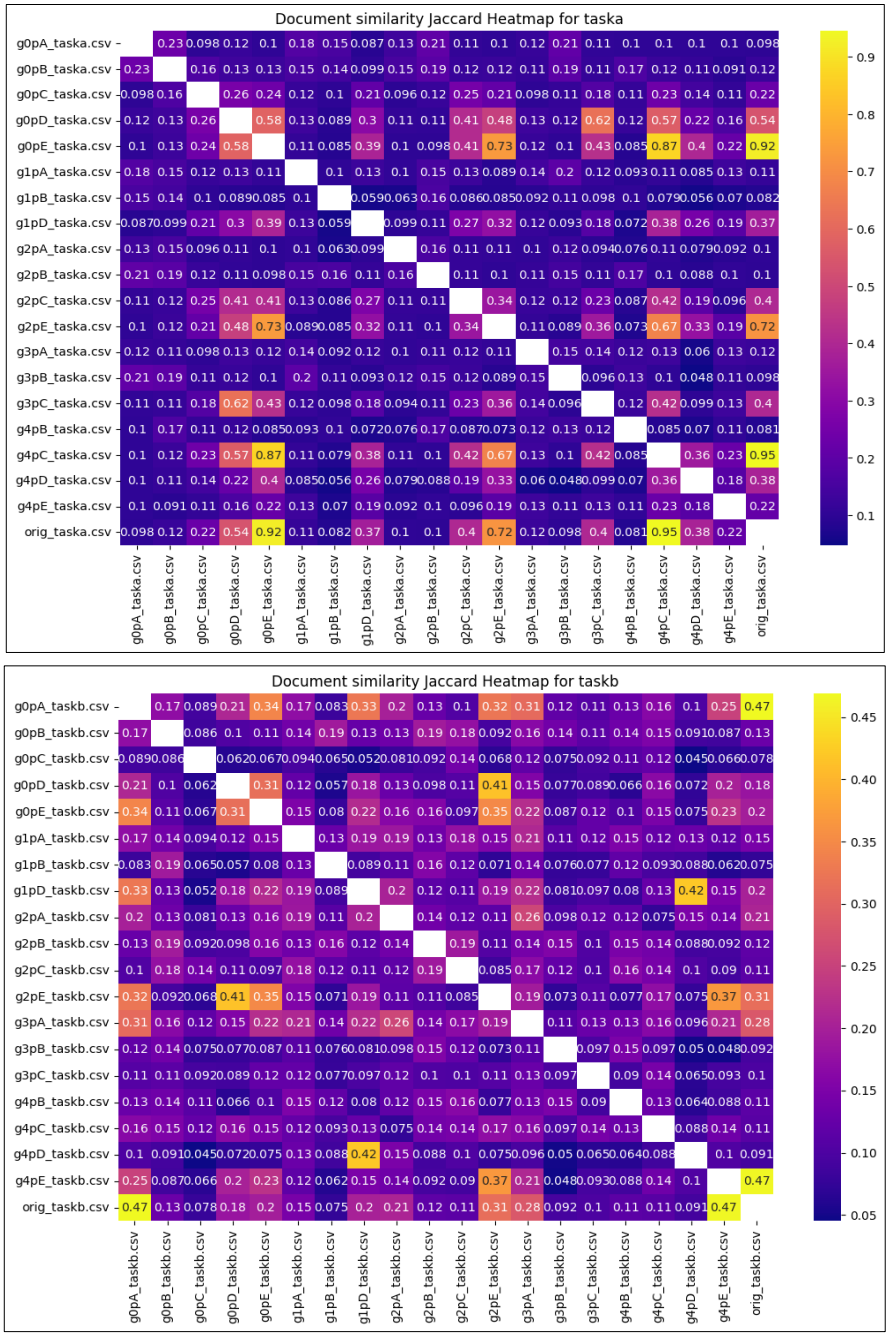
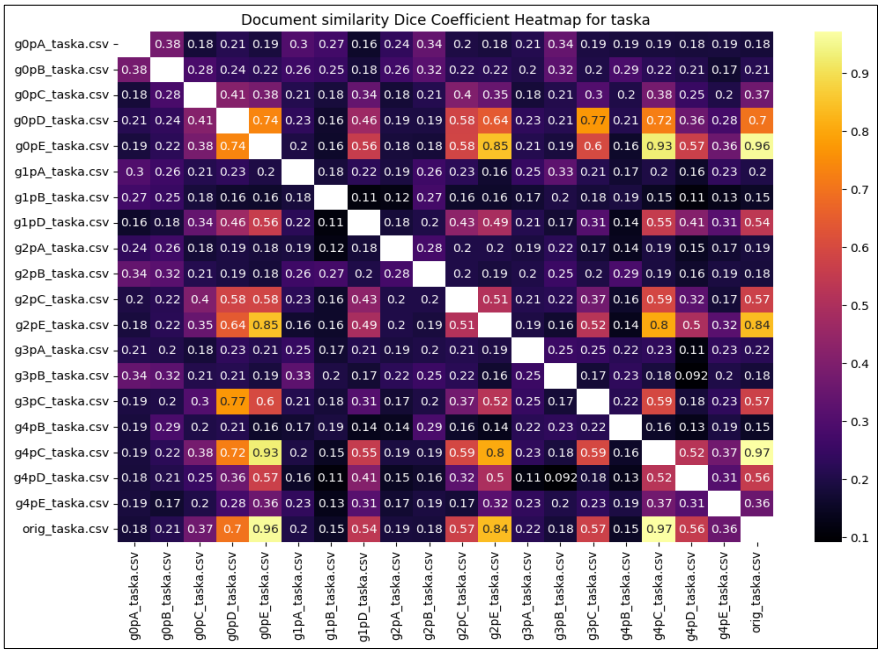
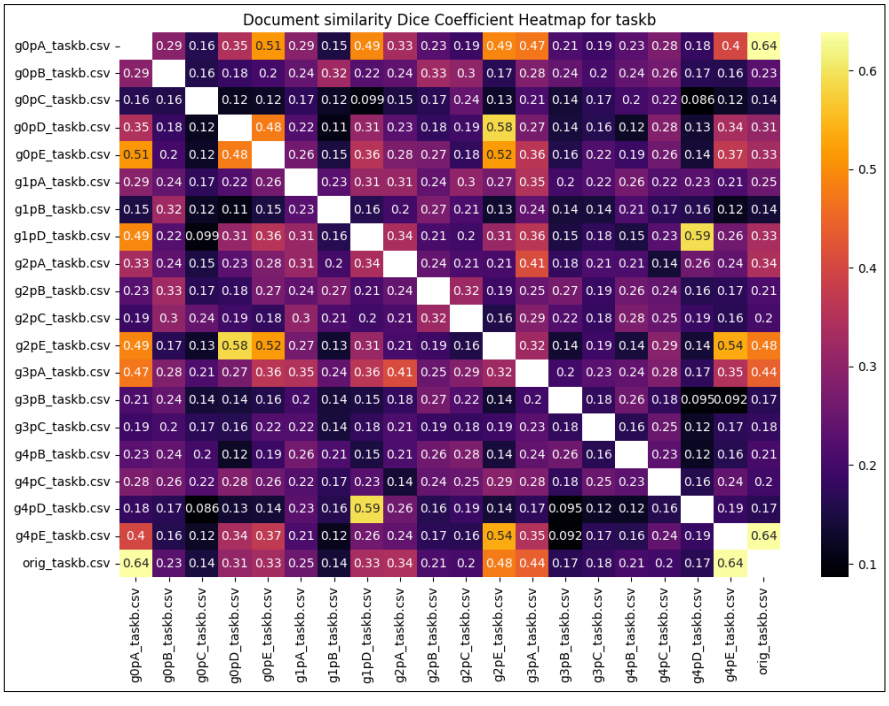


Figure 4: Dice Coefficient Heatmaps





The given heatmap represents cosine similarity value, of each file belonging to 'taska' ('taska' is a specific topic, out of the five topics as discussed in the dataset description, from which the corpus is built) by comparing each file to each other, apart from itself. But for this research we only consider those cosine similarity values that are resulting out of comparison of all the 19 copied files (g0pA_taska.csv, g0pB_taska.csv…etc.) in 'taska' to the original file original 'taska' file 'orig_taska.csv' (which contains the original answer of the topic based on which the question was asked), And we repeat the exact same process for all the remaining 4 topics - 'taskb', 'taskc', taskd', 'taske'. And we carry out the same process for Jaccard Similarity and Dice Coefficient. The given heatmaps are of taska, and taskb only however we created the heatmaps for taska, taskb, taskc, taskd, taske. The given heatmaps (Figure 2, Figure 3, Figure 4), are of taska, and taskb only however we created the heatmaps for taska, taskb, taskc, taskd, taske.

3.5 Problems faced in Lazy classifier

1. Lazy classifier did not work on textual data and was giving 0.00 output in every case when we used textual data in it.

Model	Accuracy	Balance Accuracy	ROC AUC	F1 Score
AdaBoostClassifier	0.00	0.00	None	0.00
LinearSVC	0.00	0.00	None	0.00
SVC	0.00	0.00	None	0.00
SGDClassifier	0.00	0.00	None	0.00
RidgeClassifierCV	0.00	0.00	None	0.00
RidgeClassifier	0.00	0.00	None	0.00
RandomForestClassifier	0.00	0.00	None	0.00
Perceptron	0.00	0.00	None	0.00
PassiveAggressiveClassifier	0.00	0.00	None	0.00
NearestCentroid	0.00	0.00	None	0.00
LogisticRegression	0.00	0.00	None	0.00
LabelSpreading	0.00	0.00	None	0.00
BaggingClassifier	0.00	0.00	None	0.00
LabelPropagation	0.00	0.00	None	0.00
KNeighborsClassifier	0.00	0.00	None	0.00
GaussianNB	0.00	0.00	None	0.00
ExtraTreesClassifier	0.00	0.00	None	0.00
ExtraTreeClassifier	0.00	0.00	None	0.00
DummyClassifier	0.00	0.00	None	0.00
DecisionTreeClassifier	0.00	0.00	None	0.00
CategoricalNB	0.00	0.00	None	0.00
BernoulliNB	0.00	0.00	None	0.00
LGBMClassifier	0.00	0.00	None	0.00

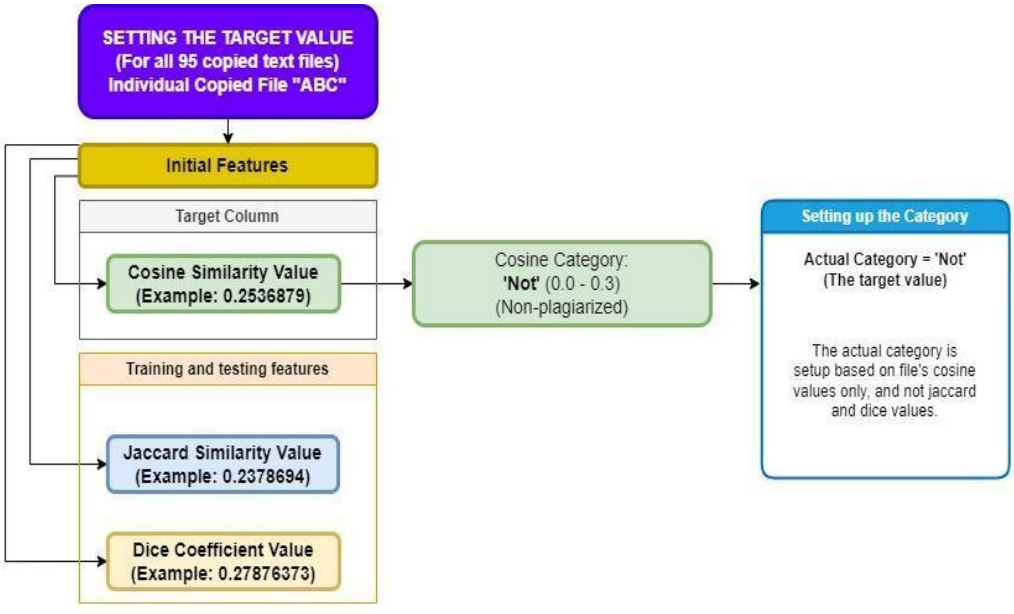
2. So we used numerical data (similarity value) that was derived from cosine similarity, jaccard and dice coefficient, of the textual data - comparison between each of the 95 copied files with their respective original 5 files.
3. First we only fed the data derived from cosine similarity,
4. using 0.5 as a threshold we divided the data between plagiarized and non plagiarized (meaning <0.5 will be non plagiarized and >0.5 will be plagiarized) which resulted in unrealistic scores of 1.00 for every recommended algorithm because dataset was heavily imbalanced as most samples were having a plagiarism label of 0 (not plagiarized) as the similarity scores were always in decimal values (eg - 0.34, 0.01, 0.234). This means that a simple classifier that always predicts 0 would already achieve a high accuracy score.
5. So we we created 20 categories of plagiarism, instead of just 2 (0.0-0.5 and 0.5 -1.0), we created 20 categories of data for a finer evaluation by the lazy classifier,
6. with only cosine similarity as a feature it was a very simple dataset which could have been resulting in overfitting by many models in the lazy classifier.
7. So now we used all three similarity values - cosine similarity, jaccard and dice coefficient as features to prevent overfitting and now the lazy classifier started showing accurate scores.

Model	Accuracy	Balanced Accuracy	ROC AUC	F1 Score
LinearDiscriminantAnalysis	0.95	0.95	None	0.95
DecisionTreeClassifier	0.95	0.95	None	0.95
BaggingClassifier	0.95	0.95	None	0.95
GaussianNB	0.84	0.88	None	0.84
AdaBoostClassifier	0.79	0.80	None	0.71
NearestCentroid	0.74	0.78	None	0.74
RandomForestClassifier	0.74	0.70	None	0.76
ExtraTreeClassifier	0.68	0.69	None	0.71
LGBMClassifier	0.74	0.68	None	0.72
LabelPropagation	0.74	0.68	None	0.71
ExtraTreesClassifier	0.68	0.66	None	0.69
LabelSpreading	0.68	0.58	None	0.64
KNeighboursClassifier	0.58	0.43	None	0.55
SVC	0.47	0.38	None	0.42
LinearSVC	0.26	0.26	None	0.28
PassiveAggressiveClassifier	0.32	0.20	None	0.17
LogisticRegression	0.32	0.20	None	0.30
Perceptron	0.05	0.10	None	0.03
RidgeClassifier	0.05	0.10	None	0.03
RidgeClassifierCV	0.05	0.10	None	0.03
SGDClassifier	0.26	0.10	None	0.15
BernoulliNB	0.00	0.00	None	0.00
CalibratedClassifierCV	0.00	0.00	None	0.00

3.6 Training and Testing

3.6.1 Strategy 1: cosine based target 'Category'

Figure 5: Cosine Based Target - 'Category'

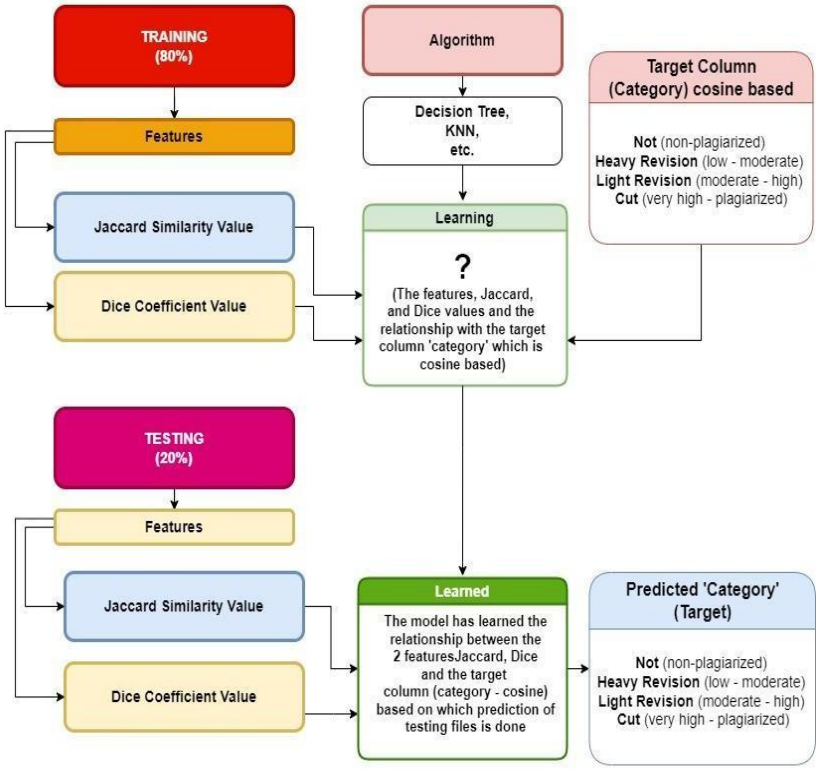


1. We categorized all files by fixing 4 categories based on cosine similarity values (Figure 5),
   - "non" (non plagiarized) from 0 to 0.3,
   - "heavy revision" (heavily revised texts, containing moderate or low plagiarism) from 0.3 to 0.6
   - "light revision" (less revised, containing high plagiarism), from 0.6 to 0.85
   - "cut" (nearly copied, plagiarized) from 0.85 to 1

   Why cosine? We are using cosine similarity value of the files to set the level of plagiarism (assumed), because on a granular level, cosine similarity focuses on directionality as well as the term frequency, whereas Dice and Jaccard focus more on the overlap of common terms without direction of usage. Cosine similarity is like comparing the angle between two arrows whereas jaccard and dice try to find common items between two baskets, so cosine with angle measurement and tf-idf weighting can be more precise in detecting levels of plagiarism. For additional information refer to work (Afzali & Kumar 2017).

2. Then we split the data of the 95 files into 76 for training and 19 for testing.
3. In the training the data we initially use features - the Jaccard Coefficient and Dice Coefficient Similarity value, and the target column 'Category' of the files (which we generated through cosine as the actual category of plagiarism)
4. We feed this data to the algorithms recommended by the lazy classifier to learn about the relationship between Jaccard and Dice Similarity and the corresponding category of plagiarism (Figure 6).
5. Then in testing we use the now learnt algorithm (which has learnt or developed some kind of an idea about the relationship between Jaccard and Dice Similarity and the corresponding category of plagiarism) to predict the category of the files in the testing data by using Jaccard and Dice similarity of the test files.

Figure 6: Training and Testing using Jaccard and Dice Coefficient

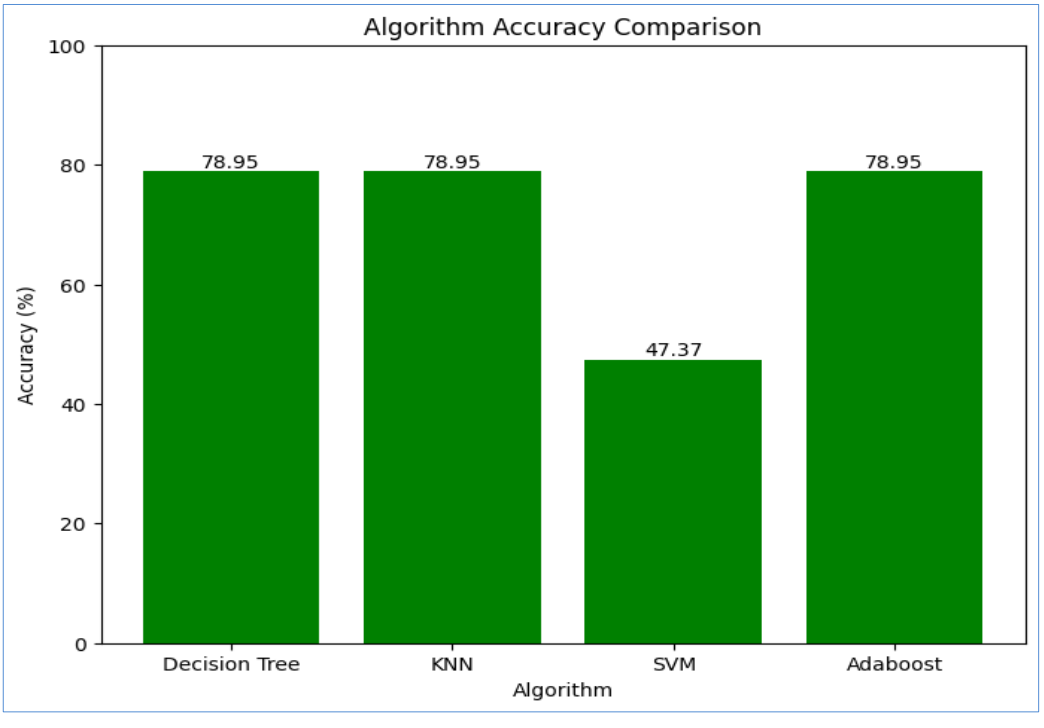


6. We compare the predicted Categories with actual Categories to take out the accuracy of our model.

7. the linear discriminant analysis gives us an accuracy of 84.21%, DecisionTreeClassifier and Bagging Classifier gave us an accuracy score of 78.9% (Figure 7) each but this is not enough, so we need to
   - Extract more features from our text data set, and use it in training of the model so we can make even more accurate predictions
   - Do hyperparameter tuning, or change strategy for training and testing

3.6.1.1 Results

Figure 7: Algorithm Accuracy Comparison



3.6.2 Strategy 2: Voting

- Earlier we set the actual categories (target value) of our files based on the cosine value ranges.

- Then we chose the algorithms recommended (Linear Discriminant Analysis, Decision Tree, Gaussian Naive Bayes, Bagging Classifier) by the lazy classifier and some other widely used algorithms like SVM and Random Forest for creating our model.
- Using Dice and jaccard values as features and the cosine based category as the target we trained the algorithm on 76 of the 95 files for training.
- After training the algorithm, we used it and tested it by giving features of jaccard and dice coefficients values of each of the 19 testing files and made it predict the category.

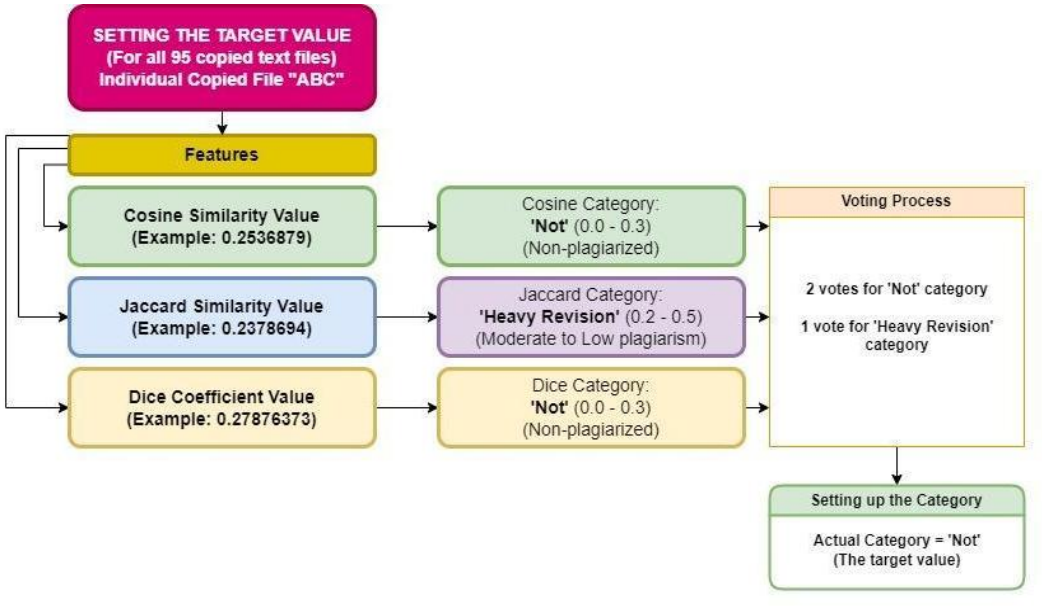
However, out of low accuracy scores, we needed to enhance our model and the way we trained our algorithm.

Table 1: Categorization Table

Category	Cosine Scale	Jaccard Scale	Dice Scale
"Not" - (non plagiarized)	0.0 - 0.30	0.0 - 0.20	0.0 - 0.30
"heavy revision" (low to moderate plagiarism)	0.31 - 0.60	0.21 - 0.50	0.31 - 0.6
"light revision" (moderate to high plagiarism)	0.61 - 0.850	0.51 - 0.750	0.61 - 0.850
"Cut" (near cut/copied or plagiarized)	0.851 - 1.00	0.751 - 1.00	0.851 - 1.00

Note; in the statistical analysis the mean values based on jaccard is approx 0.1 less than both dice and cosine similarity and so the categorization scales for jaccard is 0.1 less for all categories.

Figure 8: Voting Based target category



Furthermore, the target value could not simply be set up based on any one feature alone like we did above by taking cosine values and setting up the category (target) based on cosine because, although there is proof that cosine performs better, than Jaccard, and Dice (refer work (Afzali & Kumar, 2017)), we wanted to create a model where all three similarity measures were given weightage in both a) setting up the target category and b) training and testing.

In this new strategy to enhance our model, for each copied textual file we had 3 primary features:

- Cosine Similarity
- Jaccard Similarity
- Dice Similarity

Now we created three categories for each file based on these three values for each file (Table 1). For Eg, if a file 'ABC' has the following properties

- Cosine similarity = 0.065202536
- Jaccard similarity = 0.098360656
- Dice similarity = 0.179104478

Then we will fix the Plagiarism category for each

- Cosine Category = 'not' (non plagiarized)
- Jaccard Similarity = 'not' (non plagiarized)
- Dice Category = 'not' (non plagiarized)

Now since all three are categorized as 'not', we fix the actual category of the text file as 'not'. Meaning, the actual category that we are setting for the given Copied textual file in relation to its original file is category 'not' - it is not plagiarized or not similar to the original text file.

NOTE: If there was a copied text file which had categories

- Cosine - 'not'
- Jaccard - 'heavy revision'
- Dice - 'not'

Based on its respective cosine (say 0.2536879), jaccard (say 0.2378694), and dice (say 0.27876373) values then the actual category of that file is set to be 'not' (non plagiarized) since 2 features say 'not' and 1 says 'heavy revision' (based on voting)

So now we set the actual categories 'Category' of each copied textual file through the process specified above **(**Figure 8). This, 'Category' is our target value (which will be predicted). Or the Actual category of plagiarism that we are setting up for this file based on the values of it's features. We do this for all the copied text files in our dataset.

3.7 Training

After we are done setting up the "Category" which is assumed to be the actual category of plagiarism (target), now we start the training process.

Note: When we used this strategy using 80:20 the model was overfitting, with most of the algorithms, so we wanted more data in the testing, and that's why we changed the ratio of data from 80:20 to 60:40.

The data this time is split in the ratio 60:40 (approx) between training and testing, Each file in the dataset has features:

- Cosine similarity Value
- Jaccard Similarity Value
- Dice Similarity Value

And the target column 'Category'. Which is the actual category set up for that file.

Using these three features from the files present in the training data set, and the Target Column 'Category' We train our model using a classification algorithm recommended by lazy classifier.

How does the training process work?

The training process here will work, by feeding the algorithm the 3 feature values, and the target column 'Category' of each file in the training dataset.

This will help the algorithm *learn by understanding the relationship between the 3 feature columns i.e Cosine, Jaccard and Dice Similarity Value and the target column 'Category'* (containing the category/level of plagiarism of text) for all files in the training dataset.

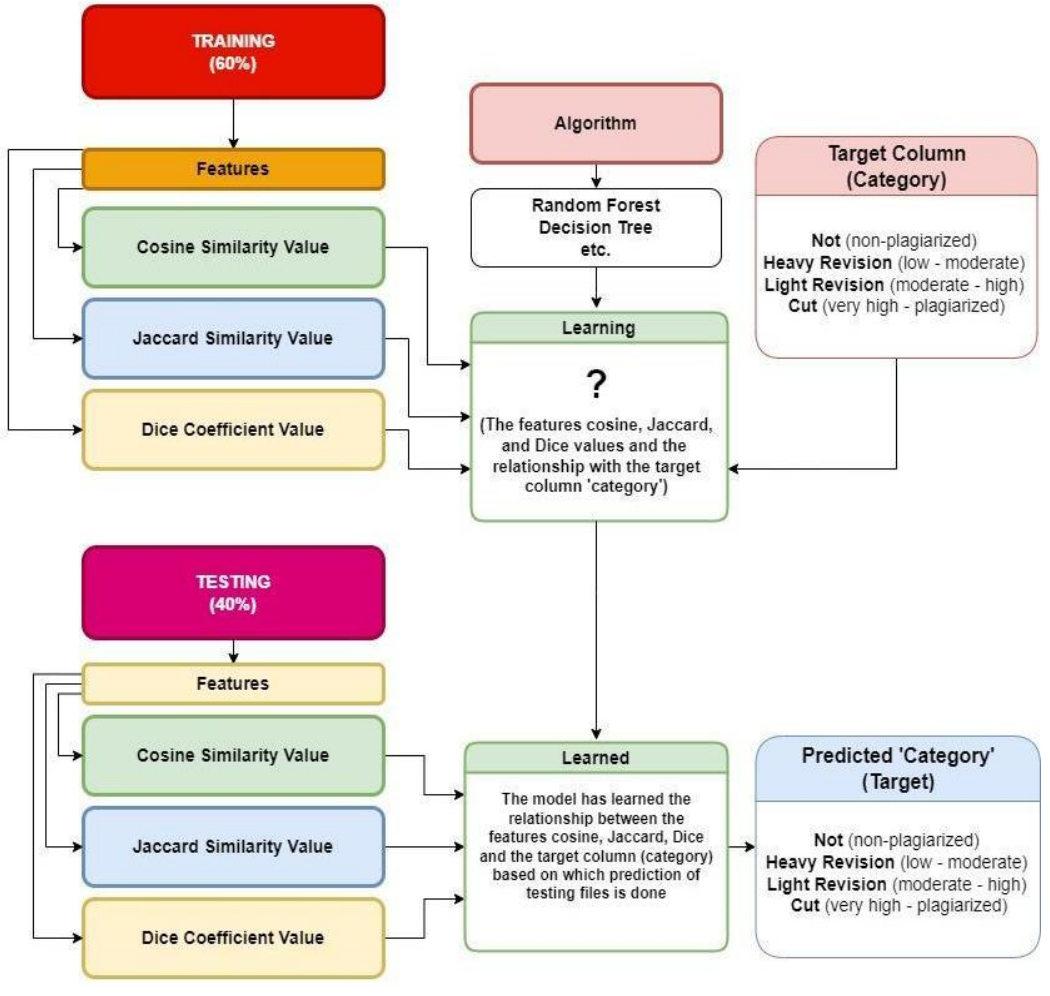
```
# Load the CSV file
df = pd.read_csv ('/content/drive/MyDrive/similarity
csv/modified_merged_categorized_plagiarism2 - all.csv')

# Define the feature columns
X = df[['Cosine Similarity', 'Jaccard Similarity', 'Dice Similarity']]

# Split the data into training and testing sets
X_train, X_test = train_test_split (X, test_size=0.4, random_state=42)

# Define the target variable
y = df['Category']

# Split the target variable into training and testing sets
y_train = y.iloc[X_train.index]
y_test = y.iloc[X_test.index]
```

This learning process develops some kind of an idea, regarding which categories correspond to which ranges of cosine, jaccard and dice similarity, the underlying pattern between the 3 numerical features and the target column 'Category'

Note that This is the training data, and the algorithm is only looking at the 'Category' and the features of files in the training data.

Meaning that during the time of training the model does not possess any knowledge about the actual category (target column) of the files in the testing data.

Figure 9: Training and Testing using Cosine, Jaccard and Dice Coefficient



3.8 Testing

Now that the model has learnt the underlying pattern or the relationship between the three features cosine, jaccard, and dice similarity values and the target column 'Category' from the training data, We begin the testing.

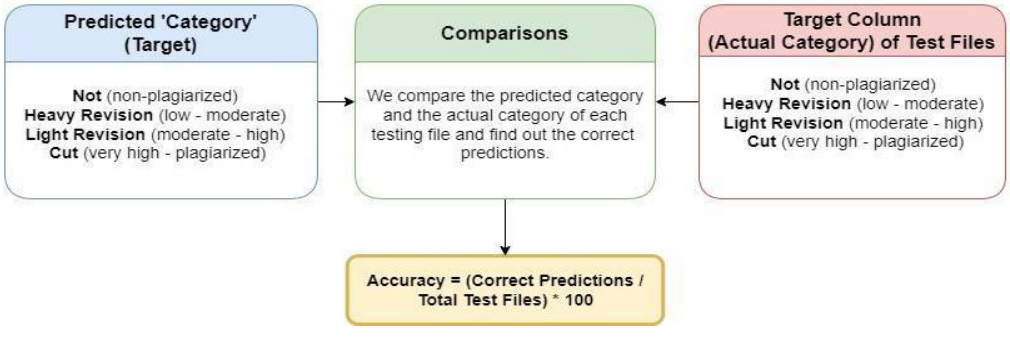
Just like training had two components - the features and the target column 'Category' (which is the level of plagiarism)

In, testing there are two components, but different

First component is the Features of the testing data set, which are the cosine, jaccard and dice similarity values and the second component is the Predicted 'Category'. In training we used features and the target Column 'Category' to train the model, and now. We use the features of the files in the testing dataset, along with the model that now has undergone training. So when the model sees a Particular testing file's feature values i.e - cosine, jaccard and dice, It will be able to predict the category to the degree to which it possesses knowledge about the relationship between the values (value set of Cosine, Jaccard, and Dice together) that are similar and the 'category' (which was gained through training). Using this learnt Model, we predict the category of plagiarism of all the files in the testing data (Figure 9).

3.9 Accuracy

Figure 10: Method for accuracy score



Finally we derive the accuracy score of our model by comparing the predicted categories of all the testing files with the testing file's actual category (Figure 10) which is the target column 'Category'. For example - if testing data contains (say) 19 files, and When we compare the predicted category, with the actual category for each file across the entire testing dataset, we find that 16 out of 19 files in the data set have been accurately

predicted. This means that the accuracy score of our model will be (16/19) * 100 = 84.2. This would imply that the accuracy score of the model will be 84.2%.

3.9.1 Results

Figure 11: Algorithm Accuracy Comparison

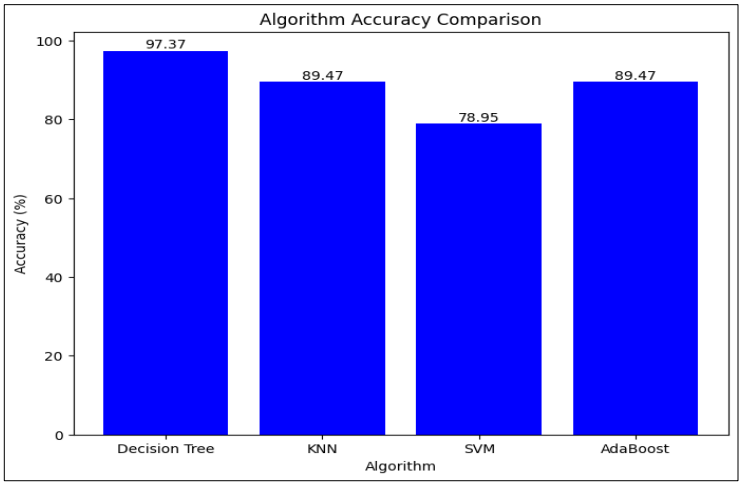


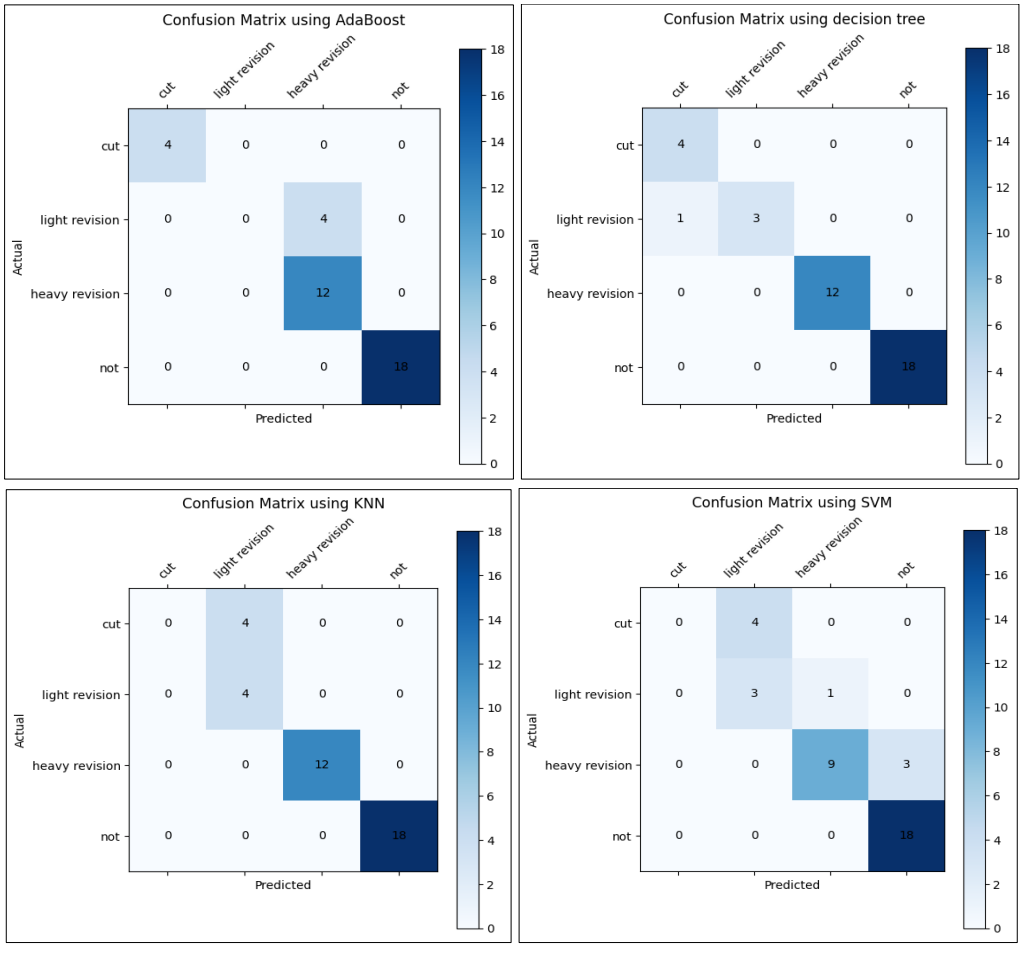
Table 2: Accuracy of Model using Different Classifiers under the Two Approaches

Machine Learning Algorithm	Strategy 1	Strategy 2 (improved)
Decision Tree	78.95%	97.31%
K-Nearest Neighbor	78.95%	89.47%
Support Vector Machine (SVM)	47.37%	78.95%
ADAboost Classifier	78.95%	89.47%

Table 3: Precision and Recall and F1-Score (Before and After)

Algorithms	Precision		Recall		F1-Score	
	Strategy 1	Strategy 2	Strategy 1	Strategy 2	Strategy 1	Strategy 2
Decision tree	0.66	0.98	0.62	0.97	0.64	0.97
K nearest Neighbor (KNN)	0.77	0.84	0.75	0.89	0.75	0.86
Support Vector Machine (SVM)	0.60	0.74	0.69	0.79	0.64	0.75
ADAboost Classifier	0.72	0.82	0.69	0.89	0.70	0.85

Figure 12: Confusion Matrix for Classifiers in Strategy 2



4.0 Conclusions

In this research we have tried to build a textual plagiarism detection model by using Machine Learning Algorithms and 3 similarity measures - cosine, jaccard and Dice coefficient values as features. In the first strategy we fix the category or level of plagiarism based on cosine values and train the model on Jaccard and Dice coefficient values in which maximum accuracy score was 78.95% (Table 2). In this we divide the dataset in 80:20 ratio for training and testing.

The Second strategy is an improvement and enhancement of strategy 1 (Figure 11, Figure 12), where we set the category of plagiarism through voting, instead of just cosine similarity, and then test based on all three features, giving equal weightage to all features in setting up the target, as well as in training and in testing. The second strategy gives us a higher accuracy score (Table 2), with a testing dataset twice the size of the testing dataset used in the first strategy. Here we divided the dataset in a 60:40 ratio for training and testing, in which all the machine learning classifiers show significant improvement compared to the prior strategy (Table 3). This model in the second strategy using cosine, jaccard and dice similarity values gives us another way of finding out text based plagiarism with a very high accuracy. Shows that cosine similarity, jaccard coefficient and Dice coefficient can be combined and can be used to detect plagiarism in various academic documents, assignments, and theses.

5.0 Future Work and Opportunities

The dataset used in the research is big but the number of files that belong to the category 'cut' (very high or copied) has relatively lesser number of textual files compared to other three 'not', 'heavy revision', 'light revision', meaning the model will struggle sometimes in differentiating between the text that belongs to the category 'cut' and the category immediately below it which is 'light revision' (mid - high plagiarism), where the the feature values are just on the boundary value that separates these two categories. This means data (text files) can be added for the 'cut' category.

References

Afzali, M. & Kumar, S. (2017). Comparative analysis of various similarity measures for finding similarity of two documents. *International Journal of Database Theory and Application, 10*(2), 23-30.

Alfikri, Z. F. & Purwarianti, A. (2014). Detailed analysis of extrinsic plagiarism detection system using machine learning approach (Naive Bayes and SVM). *TELKOMNIKA Indonesian Journal of Electrical Engineering, 12*(11), 7884-7894. Retrieved from https://doi.org/10.11591/telkomnika.v12i11.4352

Ali, A. & Taqa, A. Y. (2022). Analytical study of traditional and intelligent textual plagiarism detection approaches. *Journal of Education & Science, 31*(1), 1-10.

Bandara, U. & Wijayarathna, G. (2011). A machine learning-based tool for source code plagiarism detection. *International Journal of Machine Learning and Computing, 1* (4), 337-341. Retrieved from https://doi.org/10.7763/IJMLC.2011.V1.65

Chitra, A. & Rajkumar, A. (2016). Plagiarism detection using machine learning-based paraphrase recognizer. *Journal of Intelligent Systems, 25*(3), 351-359. Retrieved from https://doi.org/10.1515/jisys-2015-0167

Clough, P. & Stevenson, M. (2009). Creating a corpus of plagiarised academic texts. In *Proceedings of Corpus Linguistics Conference, CL'09 (to appear).* Retrieved from https://www.researchgate.net/publication/228527079_Creating_a_Corpus_of_Plagiarised_Academic_Texts

El-Rashidy, M.A., Mohamed, R.G., El-Fishawy, N.A. & Shouman, M.A., (2022). Reliable plagiarism detection system based on deep learning approaches. *Neural Computing and Applications, 34*(21), 18837-18858.

Hunt, E., Janamsetty, R., Kinares, C., Koh, C., Sanchez, A., Zhan, F., Ozdemir, M., Waseem, S., Yolcu, O., Dahal, B. & Zhan, J. (2019). Machine learning models for paraphrase identification and its applications on plagiarism detection. In *2019 IEEE International Conference on Big Knowledge (ICBK)* (pp. 97-104). IEEE. Retrieved from https://doi.org/10.1109/ICBK.2019.00021

Husain, M. I., Khan, S. & Ahmad, M. (2024). Algorithm analysis using machine learning in plagiarism detection at universities.

Kholodna, N., Makarov, O., Kholodny, V. & Pochepkina, A. (2022). Machine learning model for paraphrases detection based on text content pair binary classification. Retrieved from https://ceur-ws.org/Vol-3312/paper23.pdf

Pereira, C., Iyer, S. & Raut, C. (2017). Recommendation system based on cosine similarity algorithm. *International Journal of Recent Trends in Engineering Research, 3*(9), 2455-1457. Retrieved from https://doi.org/10.13140/RG.2.2.13587.71206

Rogozin, A., Medvedeva, M. & Ford, V. (2019). Vectorization of documents and analysis of their identity using a neural network. *CEUR Workshop Proceedings, 2562*, 1-9. Retrieved from https://ceur-ws.org/Vol-2562/

Rosu, R., Lungu, I. & Smarandache, F. (2020). NLP-based deep learning approach for plagiarism detection. *International Journal of User-System Interaction, 13*(1), 48-60. Retrieved from https://doi.org/10.4018/IJUSI.2020010104

Sabri, T., El Beggar, O. & Kissi, M. (2022). Comparative study of Arabic text classification using feature vectorization methods. *Procedia Computer Science, 198*, 269-275. Retrieved from https://doi.org/10.1016/j.procs.2022.01.038

Sadhin, I. H., Hassan, T. & Nayim, M. A. M. (2024). Plagiarism detection using artificial intelligence. *International Journal of Computer and Information System (IJCIS), 5*(2), 102-108.

Saeed, A. A. M. (2023). *Designing and implementing intelligent textual plagiarism detection models* (Doctoral dissertation). College of Computer Science and Mathematics, University of Mosul, Nineveh, Iraq.

Sahu, M. (2016). Plagiarism detection using artificial intelligence technique in multiple files. *International Journal of Scientific and Technology Research, 5*(4), 15-18.

Sari, L. P. (2023). Cosine similarity-based plagiarism detection on electronic documents. *Journal of Computer Science Application and Engineering (JOSAPEN), 1*(2), 44-48.

Shahmirzadi, O., Lugowski, A., & Younge, K. (2019). Text similarity in vector space models: A comparative study. In *2019 18th IEEE International Conference on Machine Learning and Applications (ICMLA)* (pp. 89-96). IEEE. Retrieved from https://doi.org/10.1109/ICMLA.2019.00025

Subroto, I. M. I. & Selamat, A. (2014). Plagiarism detection through internet using hybrid artificial neural network and support vector machine. *TELKOMNIKA (Telecommunication Computing Electronics and Control), 12*(1), 209-218. Retrieved from https://doi.org/10.11591/telkomnika.v12i1.3489

Tessari, F. & Hogan, N. (2024). Surpassing cosine similarity for multidimensional comparisons: Dimension insensitive Euclidean metric (DIEM). *arXiv Preprint*. Retrieved from https://arxiv.org/abs/2407.08623

Ullah, F., Wang, J., Farhan, M., Habib, M. & Khalid, S. (2021). Software plagiarism detection in multiprogramming languages using machine learning approach. *Concurrency and Computation: Practice and Experience*, *33*(4), e5000.

Yülüce, İ. & Dalkılıç, F. (2022). Author identification with machine learning algorithms. *International Journal of Multidisciplinary Studies and Innovative Technologies*, *6*(1), 45-50.

Zahid, M.M., Abid, K., Rehman, A., Fuzail, M. & Aslam, N. (2023). An efficient machine learning approach for plagiarism detection in text documents. *Journal of Computing and Biomedical Informatics*, *4*(2), 241-248.

Zechner, M., Muhr, M., Kern, R. & Granitzer, M. (2009). External and intrinsic plagiarism detection using vector space models. In *Proceedings of the SEPLN* (Vol. 32, pp. 47-55).