Volume: 14 Issue: 02 Pages: 5353-5360(2022) ISSN: 0975-0290

Enhanced Semantic Similarity Detection of Program Code Using Siamese Neural Network

Hadiza Lawal Abba
Kofar kaura layout kastina, State, NIGERIA.
Email: hadizalawalabba@gmail.com
Abubakar Roko
Department of Computer Science, Usmanu Danfodiyo University, Sokoto, NIGERIA
Email: abroko@yahoo.com
Aminu B. Muhammad
Department of Computer Science, Usmanu Danfodiyo University, Sokoto, NIGERIA
Email: aminu.muhd@yahoo.com
Abdulgafar Usman
Ministry of Finance Economic and Development, Damaturu, Yobe State, NIGERIA
Email: abdulgafarusman80@gmail.com
Abba Almu
Department of Computer Science, Usmanu Danfodiyo University, Sokoto, NIGERIA
abba.almu@googlemail.com

ABSTRACT

Even though there are various source code plagiarism detection approaches, most of them are only concerned with lexical similarities attack with an assumption that plagiarism is only conducted by students who are not proficient in programming. However, plagiarism is often conducted not only due to student incapability but also because of bad time management. Thus, semantic similarity attacks should be detected and evaluated. This research proposes a source code semantic similarity detection approach that can detect most source code similarities by representing the source code into an Abstract Syntax Tree (AST) and evaluating similarity using a Siamese neural network. Since AST is a language-dependent feature, the SOCO dataset is selected which consists of C++ program codes. Based on the evaluation, it can be concluded that our approach is more effective than most of the existing systems for detecting source code plagiarism. The proposed strategy was implemented and an experimental study based on the AI-SOCO dataset revealed that the proposed similarity measure achieved better performance for the recommendation system in terms of precision, recall, and f1 score by 15%, 10%, and 22% respectively in the 100,000 datasets. In the future, it is suggested that the system can be improved by detecting inter-language source code similarity.

Keywords –**Source Code, Lexical plagiarism, Semantic neural network.**

Date of Submission: Jul 8, 2022 Date of Acceptance: Aug 27, 2022

1. INTRODUCTION

Plagiarism of source-code is a growing problem due to the growth of source-code repositories, and digital documents found on the Internet. Therefore, identification of source code re-use is an interesting topic from two points of view. Firstly, the industries that produces software are always looking for a way to protect their developments, thus they usually search for any sign of unauthorized use of their own blocks of source code. Secondly, in the academic field, copying programs is now a well-known habit among students. Such phenomena is also motivated due to the vast number of open source codes available on the internet today and some proposed restructure approaches [26]. Probably every instructor of a programming course has been concerned about possible plagiarism in the program solutions turned in by students. Consequently, source code re-use detection has become an important research topic, motivating different groups of researchers [1], [2], [3], [4], [5], [6], [7], [8], [9], [10]. To build automatic systems to detect software plagiarism. For example, [3] was able to detect source code plagiarism using machine learning technique by transforming source code into abstract syntax trees and then split up the tree into functions, the tree for each function is considered a document. This document collection is fed to an SVM package using a kernel that operates on tree structured data with a given author and a classifier was trained with source code from two authors, and is then able to predict which of the two authors plagiarized a new function.[8]Also tried to detect source code plagiarism based on Abstract Syntax Tree (AST) but the system was not scalable due to lots of pairwise comparison of tree structures. Therefore, [10] approached the problem of scalability in an information retrieval perspective. While the IR approach is efficient, it is essentially unsupervised in nature. Therefore, to improve the effectiveness of their work, they applied a supervised classifier (trained on features extracted from sample plagiarized source code pairs) on the top ranked retrieved documents. Their work evaluated a set of three different types of features in order to determine the similarity between code sources. The features are: 1) lexical (character n-grams), 2) structural (function names and parameter names and types), and 3)

Volume: 14 Issue: 02 Pages: 5353-5360(2022) ISSN: 0975-0290

stylistic (the number of lines of code, the number of white spaces, the number of tabulations, the number of empty lines, the number of defined functions, average word length, the number of upper case letters, the number of lower case letters, the number of underscores, vocabulary size, and the lexical richness). This combination has shown important aspects, with acceptable results, for determining plagiarism between pairs of Java source codes. However, the combination of these features is more oriented in detecting plagiarism based on the lexical similarities between documents.

After evaluating the stated features, the system compute their similarity using the cosine similarity measure. This cosine similarity algorithm calculates the similarity between two text documents by counting all 3-grams available in the documents and stores them in an M dimensional vector, where M is the number of unique N-grams found in the passage. Cosine similarity algorithm uses these vectors to compare documents. Generally, cosine similarity measure has better recall and precision than most of other text similarity measures like Pearson correlation coefficient (PCC), Jaccard or mean square difference (MSD) [11]. However, this measure involves simultaneous comparison of a single document to all other documents and the vectorization step might depend on whole document corpus storing the vectored form of each document resulting to the storage of large amount of data. Also, due to excessive comparison step by the cosine similarity algorithm, the system has high complexity of O (n^2). Cosine similarity has a very effective drawback; it computes similarity based on frequency of words therefore it is unable to detect semantic similarities between documents; producing inaccurate results. However the AST representation of the source code documents ignores the existence of functions which a very essential user input.

To address the above mentioned problems, in this study we propose an enhanced plagiarism detection method called AST_Neural System,where some additional terms will be added as an AST node instead of a table [24], because a tree captures aspect of source code that are inherent to the programmer's input more than the lexical features of a particular programming language. The system employs a model that captures both lexical and semantic similarities between documents in order to produce more accurate results.

2. RELATED WORKS

A. Alex. [1] Developed a source code plagiarism detection system called MOSS. MOSS (measure of software similarity) is based on a string matching algorithm that functions by dividing programs into k-grams, where k is a contiguous substring of length k. each k-gram is hashed and MOSS selects a subset of these hash values as the program's fingerprints. Similarity is determined by the number of fingerprints shared by the programs, i.e. the more fingerprints they shared the more similar they are. For each pair of source code fragments detected, the results summary includes the number of tokens matched, the number of lines matched and the percentage of source code overlap between the detected file pairs. MOSS is a wonderful system and a major advancement in the plagiarism detection area. However, the system can only detect the copy and paste plagiarism effectively and produce much false retrieves when dealing with more complicated plagiarism cases.

I. Baxter, A. Yahin, L. Moura, M. Anna, and L. Bier. [2] Presented a simple and practical methods for detecting exact and near miss clones by using AST in program source code. The method was based on hashing, firstly the source code was parsed and an AST was produced for it and then three main algorithms were applied to find clones. The first algorithm (basic algorithm) was to detect sub tree clones, the second (sequence detection algorithm) was to detect variable size sequence of sub-tree clones and the third (last algorithm) looks for more complex near miss clones by attempting to generalize combinations of other clones. For the algorithms to find sub tree clones to function, every sub tree is compared to every other sub tree for equality. The method is straight forward to implement but the algorithms used performs better on a dataflow graph than trees. This may likely results to high number of false positive retrieves when detecting near miss clones. B. N. Pellin [3] presented a technique for detecting authorship of a source code. He used machine learning technique to accomplish the task by transforming source code into abstract syntax trees and then split up the tree into function. The tree for each function is considered a document, with a given author. This collection is fed to an SVM package using a kernel that operates on tree structured data. The classifier is trained with source code from two authors, and is then able to prediction which of the two authors wrote a new function. The method was able to achieve between 67% and 88% classification accuracy over the set of programs examined. However, the method is highly vulnerable to manipulation of the source code, an advanced source code translator or obfuscator could destroy the patterns that their classifier uses to identify authors and also it requires that you know the set of possible authors. J. Son, S. Park, and S. Park [5] proposed a plagiarism detection system that uses parse tree kernels. The role of the parse tree kernels in the system is to handle the structural information within source programs and to measure the similarity between parse trees extracted. The system performs 100% accurate for a simple attack and it is not affected by structural attack. The system is independent of programming languages. Due to the structure of copied programs which includes a lot of abundant garbage, a lot of plagiarism detection systems fail in detecting plagiarism. However, the system is liable to present false positive outcomes because the similarity values of the parse tree kernels increases too fast to handle and the value of the kernels between two different trees is typically much smaller than the value between same trees. C. Liu, C. Chen, J. Han and P. S. Yu. [6] Developed a new software plagiarism detection tool called GPlag by mining program dependence graphs (PDGs). GPlag is a PDG-based scalable detection tool. The tool works using a PDG-based plagiarism detection tool, it takes an original program and a suspected

Volume: 14 Issue: 02 Pages: 5353-5360(2022) ISSN: 0975-0290

plagiarized program as input. And outputs a set of PDG pairs that are regarded as involving plagiarism. Experiments showed that GPlag is both effective and efficient. However people need to examine the returned PDG pairs to confirm plagiarism and/or eliminate false positives, which makes the tool plagiarism manual. D. Zou, W. Long, and Z. Ling.[4] described a cluster-based plagiarism detection method to detect plagiarism in the network engineering related courses. The method consist of 3 steps: the first step is the preselect step, which is to find a small list of candidate documents from the source document set which may contain the plagiarized content. The next step is to compare the suspicious document with each candidate document to get the copied part from the suspicious document, this step is called locating. The last step is called post processing, which is to discard some fragments without plagiarism from the end result. The method was tested on both the training and testing set for PAN-09 and proved to be effective, but due to its multiple operating steps, it consumes much execution time. Therefore, the method is said to have very high big O complexity. E. Flores, A. Barro ´n-Ceden ˜o, P. Rosso, and L. Moreno. [11] Proposed a simple approach to the detection of cross-language source code re-use. Their experiment was based on character 3-grams comparison and where able to achieve acceptable result when comments were ignored. However, due to the massive number of similar keywords in a programming language, the character n-grams results to an over estimation of lexical similarities producing inaccurate results.

To detect plagiarism, most existing commercial software adopt methods like sentence matching or keyword matching to detect plagiarism; such methods are learnt to be ineffective. Therefore, [7] introduced a detection method to solve the “copy paste” and “paraphrasing” type plagiarisms. He proposed the cosine metric factor to illustrate the relevance among documents. Data was first preprocessed by removing stop words to eliminate the relevance among unwanted words. The processed data was later stemmed to get the original form of words in the documents. The cosine metric factor was then proposed to illustrate the relevance among documents. However, the method can only work when the correct source is provided, improper edition of the reference makes the method inefficient. Z. Duric, and D. Gasevic. [8] Designed and developed a source code similarity detection system (SCSDS) in order to solve the problem of structural modification source code which is a confusing factor for most similarity detection systems, making them produce inefficient results. The system consist of two parts, SCSDS core and SCSDS frontend. The first part SCSDS core consist of four modules, the preprocessing module; which perform preprocessing on the source document and forward its output to the tokenized module. The tokenized module performs the tokenization, which is a similarity determination between file documents, the results of the module is sent to the exclusion module. The exclusion module removes an “exclusion token sequence” from the list of tokens obtained from the tokenization module. The resulting list of tokens will be forwarded to the comparator module. The comparator module is responsible for similarity measurement using RKR-GST comparator (which implements the RKR-GST algorithm) and the Winnowing comparator (which implements the Winnowing algorithm). The second part, that’s the SCSDS frontend is responsible for selection of source code files for comparison and displaying of similarity measurement results. The system was tested on set of java source files. It also showed promising results in terms of performance compared to JPlag. However, SCSDS has no preprocessing module for other programming languages except Java, and an efficient and easy accessible user interface is critical to wide adoption.

L. Zhang, D. Liu, Y. Li, and M. Zhong. [9] Developed a code copy detection system based on AST. The system converted the code to AST after undergoing some preprocessing stages and formatting, it was able to detect plagiarism by calculating similarity for the AST. Due to lots of pairwise comparison, the system cannot process large dataset and have relatively high complexity, also it can only perform on C programming language. S. Narayanan and S. Simi. [12] Developed an algorithm based on fingerprint approach to identify the reuse of source code in direct and indirect ways. The system used multiple phases to detect plagiarism effectively. It takes source code from the database as input, and it will pass through the code restructuring, tokenization, complexity analyzing and the similarity computation phases. It will outputs the similarity score after passing through the presented stages. The files will be compared to figure out the similarities and the plagiarism detection reliability measure is based on precision and recall. The system was tested using large datasets in C, C++, java, and c# programming languages. However, the system was only limited to the stated programming languages and ineffective in inter language similarity.

Another common approach to source code plagiarism detection is to determine the fingerprint of a source code document by making use of the word n-grams. R. Marinescu. [13] Proposed a system for code refactoring, representing different source code segments with hashes which are compared using the Winnowing algorithm. The system was able to detect any proof of cheating and it works against different level of similarity, low level similarity till high level ones. However, this approach does not consider important characteristics inherent to source code such as keywords, identifiers names, number of lines, number of terms per line, number of hapaxes etc. S. Ion and I. Bogdan. [14] Proposed a source code plagiarism detection method. The method was based on ontologies created using protégé editor. They build ontologies for each source code that is suspected of plagiarism based on the vocabulary and taxonomy of a programming language source code using protégé (a free open source ontology editor). This process is done automatically using a crawler. The crawler will read the read the code line by line from top to bottom and will create the specific individuals for each line of code and the individual ontologies created from the presented process will be compared to see the plagiarism degree.

Volume: 14 Issue: 02 Pages: 5353-5360(2022) ISSN: 0975-0290

The method is suitable for complex software plagiarism detection but have a little drawback, the system is not fully automatic. T. Ohmann and I. Rahal. [15] Presented an approach called program it yourself (PIY) which utilized K-gram-based pairwise document comparison and PAM clustering. The system achieves high plagiarism detection accuracy with far lower runtime. However, the system can only process 10,000 documents and below which makes the system highly inefficient. J. Zhao, K. Xia, Y. Fu, and B. Cui. [16]Proposed an effective plagiarism detection algorithm based on AST. The algorithm compared code based on AST, it raised the efficiency of comparison by transforming the storage format of the syntax tree twice and converting the tree-like structure in a linear list and regrouping the sub tree according to the number of sub-nodes. The algorithm also reduce mistakes by calculating the hash value of operations (subtraction, division, modulo arithmetic, etc.). It was concluded that the system; code comparison algorithm (AST-CC) can perform more efficiently than AST because of its storage form. Therefore, just like AST the system can process large data set but due to lots of pairwise comparison, the system possess relatively high complexity. N. More, A. A. Bhootra and C. A. Patel. [17] Showed a method to detect plagiarism in Java source code. Firstly, the files were uploaded and comments were removed at the initial step. A token file was created for each uploaded source code. The token files were compared, the result of each comparison is a value called percent match. If the percent match of a pair of token files is larger than the minimum value, then the comparison pair will be assumed to be plagiarized. This is done to every file in the submitted folder. The method was tested on a set of students' assignment and was proved to be effective. However, the method can only work on small collection of dataset and can only process Java files. N. Shah, S. Modha, and d. Dave. [18] Proposed plagiarism detection methods which analyze source code using 3 different representations. The method represent code in 3 views; lexical, structural and stylistic. The method tries to find similarities based on these 3 representations. Manual threshold of the two codes are being computed for each view. The approach was concluded to give sufficient results by achieving the best precision at 97.87% and recall at 84.52%. However, only 3 structures were taking into consideration and the structural similarity is of low quality. Also, due to lots of pairwise comparison, the method cannot process large dataset and has high complexity. O. Karnalim. [19] Proposed a source code plagiarism detection which can detect plagiarism at any level by utilizing low-level instructions instead of source code tokens. The author selected java as target programming language and bytecode as its low-level instruction. It was evaluated that the approach is more effective to detect most plagiarism attack types than raw source code approach. However, the method was unable to handle plagiarism attacks in object-oriented programming. O. Karnalim. [20] Proposed an expansion of the Karnalim's approach for plagiarism detection based on low-level tokens, by incorporating three contributions, which are: flow-based token weighting which reduced the number of false-positive results, an argument removal heuristic that generates more accurate linearized method content and the invoked method removal that fastened the processing time. The approach was proved to be partially effective to handle plagiarism attacks in practical environment, however it is unable to handle source code plagiarism in object oriented environment. M. Duracik, E. Kirsak, and P. Hrkut. [21] Developed a system which focuses on representing source code using AST in order to detect plagiarism. The system represents source code using hashing and characteristics vector. They carried out an experiment based on these two approaches and tried to compute the similarity of classes as well as methods in a source code dataset; which consists of 59 student submissions. They tried to minimize absolute similarity comparison (addressing the weakness of the MOSS algorithm) but their system was unable to achieve scalability and also some false positive matches were generated at lower values. However, the system was able to prove that generating vectors using AST is the best appropriate way of representing source code M. Duracik, E. Kirsak, and P. Hrkut. [22] Developed a new scalable system to detect plagiarism in a huge number of source files by applying an incremental clustering approach in order to achieve modularity and scalability. The system transformed the source code into an AST, then characteristics vectors were generated from the tree. These vectors are clustered by the system using incremental k-means algorithm and inserted into a database. Similar vectors are then searched in the database and are post-processed. A final plagiarism report that contains a similarity score and parts of matched source code snippets was generated. The system can successfully replace the MOSS system because of its scalability and ability to search for plagiarism on a much larger scale. However, adding data to the database and maintaining consistency by re-clustering is time consuming operation and also the algorithm speed was not evaluated.

O. Karnalim. [23] Extended the karnalinm's work, a low-level approach for detecting java source code plagiarism by incorporating abstract method linearization. The method was to enhance the accuracy of low-level approach in term of detecting plagiarism in object-oriented environment. It incorporated linearization to predict the content of abstract method by concatenating all method contents from its respective implementers. The method produced less false positive retrieves and provides more accurate results since it only considered semantic preserving token. Despite its accuracy, the method suffers a drawback. It cannot detect several short similar pairs which can only be detected through standard lexical token approach. D. Ganguly, G. J. F. Jones, A. Ramı´rez-de-la-Cruz, G. Ramı ´rez-de-la-Rosa, and E. Villatoro-Tello. [10] Proposed a scalable source code plagiarism method using information retrieval approach. They perform their experiment in stages, for the first stage they retrieve a ranked list of documents using the AST representation of source code. At the second stage, they employ a

Volume: 14 Issue: 02 Pages: 5353-5360(2022) ISSN: 0975-0290

supervised approach to perform a more fined grained analysis over the set of retrieved documents from the first stage. Their work was able to achieve scalability but due to the combination of features used at the supervised stage, the system was unable to produce accurate result and also the similarity measure can only detect lexical similarities.

The above literature review presented in this chapter describes different source code plagiarism detection systems. This system works based on different methods and approaches. The strengths and weaknesses of the systems were also highlighted, up to the last review. The methods were able to detect plagiarism thou mostly on a small dataset and they are likely to produce false positive results. A research gap was described which will be address in this research.

3. MATERIALS AND METHODS

Ast_neural system

To address the above problem, the following system is proposed.This section describes the proposed plagiarism detection method that improved the method described in 4.1the system presents an improved approach of the previous version of anti_bow system.

The system performs in two stages, the IR stage and the similarity evaluation stage. At the IR stage, the documents are represented as an AST and a pseudo-query was extracted from a preset number of terms from each field of a document. The selected fields are:

i. Classes: names of java were considered.
ii. Method calls: Method names with actual parameter names and types.
iii. String literals: Values of the strings
iv. Method definitions: Names of methods and formal parameter names and types
v. Package imports: names of imported packages
vi. Arrays: Constant names of array and dimension
vii. Assignment statements: Variable names and types
viii. Comment: Text inside comments
ix. Functions: Function definitions and number of function calls

The following are the specific nodes of the AST that was used, not whole source code documents is used as a pseudo-query.

The field language model (LM) was used as the term selection function to obtain representative terms from each field. Firstly, the approach introduces a new field to the AST. The addition of a field in the AST representation can guarantee an improvement at the retrieval stage. After the first k terms are retrieved instead of employing a classifier for classification, this approach employs the Siamese Neural Model (SNM) to capture deep semantic similarities from the retrieved documents from the IR stage. The first step in an SNM is the word embedding and word frequency. The source code will be split into variables, function name, operators, reserved words, constant values and others. Each of the word is mapped to its corresponding vector with a frequency. Then the second step is the source code representation, the input are code matrices gotten from the previous step. And the last step, the similarity feature is calculated based on hidden features of code snippets.

3.1 WORD EMBEDDING

Word2Vec is a widely word embedding model applied in NLP. When using word to vectors to represent code snippet, the weight of all word vectors are equally. But, we all know the contribution of each word in code is different. For source code snippets, the weight should reflect the structure of source code. In order to improve the effect of code presentation, the words of control structure should have different coefficients and multiplied by the word frequency of it. So, the vector of code snippet is represented as

$U = \{f_1 w_1, \dots, f_i w_i, \dots, f_m w_m\}$, where $w_i \in W$ is the i^{th} row of embedding matrix W, $\{f_1, f_2, \dots, f_m\}$is Tf-Idf value of each word, m is the number of words in code snippets.

3.2 CODE REPRESENTATION

This in the previous stage, we got the embedded matrix U, at this stage both the semantic and the structural materials for the input source code will be represented. This stage is the CNN model, it contains five layers: input layer, convolution layer, pooling layer, connection layer and output layer.

Input Layer: A pair of pre-trained word embedding matrixU^A, U^B are taking as input. $U^A \in R^{N_1 \times d}$, $U^B \in R^{N_2 \times d}$,where$N_1, N_2$is the number of words respectively in source code A and B, d is the vector dimension. The two code snippets are padded with 0 to have the same length $N = max\{N_1, N_2\}$. After filling with 0, the initialized matrix $U \in R^{N \times d}$.

Convolution Layer: Each kernel $K \in R^{s \times d}$ does convolution operation in the word sequence$\{v_1, v_2, \dots, v_n\}$.

$$p_i = U_i * K \quad (4.1)$$

Here, $*$ is convolution operator,$U_i = \{U_i, U_{i+1}, \dots, U_{i+s-1}\}$, that is the embedding matrix of word sequence$\{v_i, v_{i+1}, \dots, v_{i+s-1}\}$,$1 \le i \le N - S + 1$. p_iIs a real number, because the dimensions of kernel and word vector are same. $p_i = \{p_i, p_{i+1}, \dots, p_{i+s-1}\} \in R^{d_1}$, where d_1=$N - s + 1$.

Pooling Layer: Pooling (including min, max, average pooling) is commonly used to extract robust features from convolution, to reduce its dimension. A convolution layer transforms an input feature map U with d columns into a new feature map P with one column. The maximum is gotten after each max-pooling over each vector P, which can be expressed as:

$$x_i = max\{p_i\} \quad i = 1,2, \dots, N \quad (4.2)$$

Where N is the filter number that was set in convolution layer.

Volume: 14 Issue: 02 Pages: 5353-5360(2022) ISSN: 0975-0290

Connection Layer: In connection layer, each x_i is concatenated which was gotten from pooling layer, into a vector for source code A and B.

$$X = x_1 \oplus x_2, \ldots, \oplus x_M \quad (4.3)$$

Where $\oplus$ is the operation that merges two vectors into a long vector, X is a new feature map for source code.

E.g.$a \oplus b$

Using Pythagoras theorem

$$c^2 = a^2 + b^2$$

$$R^2 = a^2 + b^2$$

$$R = \sqrt{a^2 + b^2}$$

Where R is the resolution of the concatenated vector.

Output Layer: this layer computes the similarity score. The Layer targets at calculating the similarity score of each source code pair, which can be used to rank candidate code snippets to find similar ones for any source code. The similarity score of the input pair is computed by using cosine function on new feature vectors X which leverage their semantic representations and structure representations.

$$sim(X_A, X_B) = \frac{X_A . X_B}{||X_A||_2 ||X_B||_2} \quad (4.4)$$

Where · is inner product of vector X_A andX_B, where $X_A = ||X_A||_2$ and $X_B||_2$ is their 2-norm. Those code snippets with the largest similarity score will be returned as similar codes of the given one.

Algorithm 1: IR stage
Input: Dataset Consisting Of Documents (D_1,D_2, … … … … D_N)
Output: suspected documents

1. WHILE AST representation // represent source code into an AST
2. For each source code document do
3. $LM(t, f, d) = \lambda \frac{tf(t,f,d)}{len(f,d)} + (1 - \lambda) \frac{cf(t)}{cs}$ //term selection function
4. CONSTRUCT a Pseudo-query //the suspicious document is considered a query
5. RETRIEVE documents //retrieve a ranked list of documents from the dataset
6. Query_Search(Dataset) //perform a grid search of documents from the dataset
7. Return suspected documents

Algorithm 2: Similarity Evaluation stage
Input: Suspected Documents
Output: Plagiarized documents

1. For all suspected documents do
2. CONVERT word to vector
3. $U = \{f_1 . w_1 \ldots \ldots f_i . w_i \ldots \ldots f_m . w_m\}$
4. Represent semantic and structural materials
5. U^A, U^B = INPUT //U^A, U^B are pre-trained embedding matrix
6. $U^A \in R^{N_1 \times d}$, $U^B \in R^{N_2 \times d}$ //N_1, N_2 are number of words in source code A and B
7. For each Kernel in the vector $\{v_1, v_2 \ldots \ldots . v_n\}$ do
8. $p_i = U_i * K, U_i = \{U_i, U_{i+1} \ldots \ldots U_{i+s-1}\}$
9. EXTRACT robust features
10. $x_i = max\{p_i\}$ $i = 1,2, \ldots \ldots N$
11. CONCATENATE each x_i in a vector //for source code A and B
12. $X = x_1 \oplus x_2 , \ldots , \oplus x_M$
13. COMPUTE similarity
14. $sim(X_A, X_B) = \frac{X_A . X_B}{||X_A||_2 ||X_B||_2}$
15. IF $sim(X_A, X_B) \neq 0$
16. Return plagiarized document
17. Else

Return none plagiarized

4. EXPERIMENTS AND RESULTS

This section discusses the experimental setup to evaluate our proposed AST_neural system. The effectiveness of the system was evaluated using F-Score, Precision, and Recall, evaluation metrics. Precision, Recall and F-Score are used because they are of probabilistic setting which allow us to obtain more accurate values rather than estimates. These combination of metrics is also suitable on different source codes from the same dataset where competing results are obtained on the same data.

4.1 Experimental setup

In this study, python 3.9 with pandas, numpy, matplotlib, SCIKITLearn libraries were used for the implementation and evaluation of the proposed system. The AST for each Java source code was obtained with the help of ‘‘Java parser’’ which is an open source syntax parser for Java programs. Information extracted from the AST nodes was then used to construct the field representation for every document in the index. We indexed the document set using Lucene (version 4.6). A HP Stream 11 was used for the experiment over a Linux operating system.

4.2 Data set

AI-SOCO dataset were considered which contain a considerable amount of programs. The AI-SOCO dataset is essential to detection of undesirable deception of others’ content misuse or exposing the owners of some anonymous hurtful content. This is done by revealing the author of that content. This facilitates solving issues related to cheating in academic, work and open source environments. Also, it can be helpful in detecting the authors of malware software’s over the world. The dataset is composed of source codes collected from the open submissions in the Code forcesonline judge. A code force is an online judge for hosting competitive programming contests such that each contest consists of multiple problems to be solved by the participants. A Code forces participant can solve a problem by writing a solution for it using any of the available programming languages on the

Volume: 14 Issue: 02 Pages: 5353-5360(2022) ISSN: 0975-0290

website, and then submitting the solution through the website. The solution's result can be correct (accepted) or incorrect (wrong answer, time limit exceeded, etc.). The dataset consists of 100 source code documents collected from 1,000 users. So the total of source codes documents is 100,000. For each user, all collected source codes are from unique problems. The issue of scalability has been tackled. This can be obtained at the github repository. We decide to use AI-SOCO dataset because all collected source codes are dataset collection are correct, bug-free and compile-ready. For each user submission, all collected source codes are from unique problems.

4.3 Result and Discussion

4.3.1 Results

This section presents the results of the experiments for the evaluation of the enhanced plagiarism detection method compared to anti_BoW system. Several experiments were conducted to detect plagiarism. The results obtained are presented as follows:
The Table 1: bellow shows the overall result of our proposed schemes.

Table 1: Overall results

Data Size	Model	F-Score	Precision	Recall
100,000	Anti_bow	0.46	0.80	0.80
	AST_neural	0.72	0.90	0.95

4.3.2 Discussion

Fig. 2, 3 and 4 demonstrate the precision, recall and f1 score of the Ast_neural system as compared to anti_bag of words detection method. The figures show that Ast_neural system has a higher precision and recall values compared to the other plagiarism detection method. The reason for this performance is the improvement at the IR stage and also the replacement of the classification phase with the Siamese neural network which helps to capture the semantic similarities available.
In Fig. 2, it shows the precision values measured individually for both methods with the intention to observe the new method's effectiveness. The results were obtained when testing the first 100 documents retrieved from the IR stage. It was observed that the previous method has an increasing precision from the first 40 documents and it became constant once the documents are more than 40, while the enhanced method its precision increased once the number of number of documents increases. This shows that the new method shows more effective result when testing with a large dataset.
The second figure (Fig. 3) shows the recall of the two methods. From the graph, it shows that Ast_neural method outperformed the anti_bow method with 10%. The anti_bow method considered 20 documents and above to measure recall while Ast_neural method considered even lesser amount of documents. The two methods have an equal recall when considering 20 documents that is because both methods consider a number of 20 documents from the ranked list of retrieved documents from the IR stage as an initial testing set for the second stage. The anti_bow method has a relatively low F1 score of 40-50%. The graph shows that Ast_neural method has a fluctuating F1 score when considering larger documents and increase rapidly when testing around 30-40 documents.

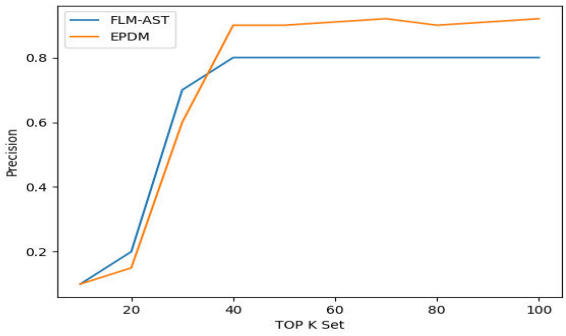

Figure 1: Precision of the two plagiarism detection methods

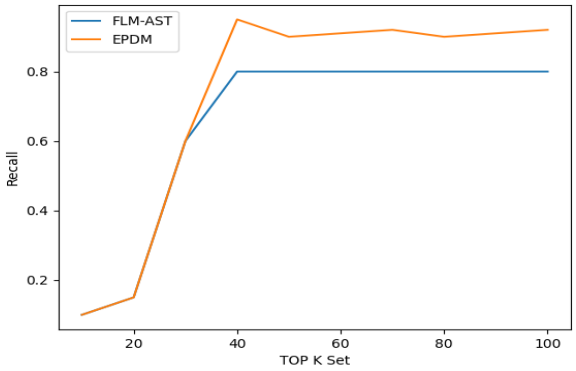

Figure 2: Recall of the two plagiarism detection methods

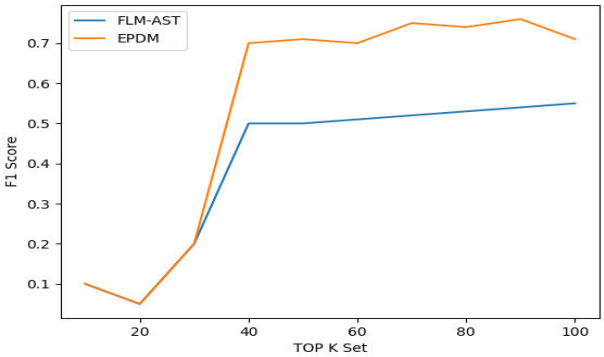

Figure 3: F1 score of the two plagiarism detection methods

5. CONCLUSION

In this study, an enhanced plagiarism detection method is proposed to produce more accurate result. Ast_neural

Volume: 14 Issue: 02 Pages: 5353-5360(2022) ISSN: 0975-0290

system introduces a new term to the term selection function in other to capture more user inputs. The method also employs a similarity evaluation model which captures deep semantic similarities between source code documents. The experimental results indicate the proposed Ast_neural system outperforms the other compared method in terms of precision, recall and f1 score.

REFERENCES

[1] A. Alex. MOSS (Measure of software similarity) plagiarism detection system 1994. Retrieved from *http:/www.cs.berkely.edu/~moss/.* University of Berkely, CA.

[2] I. Baxter, A. Yahin, L. Moura, M. Anna, and L. Bier. Clone detection using abstract syntax trees. *IEEE. Published in the Proceedings of International conference on software maintenance (ICSM'98):* 1998.pp 368-377.

[3] B. N. Pellin. Using classification techniques to determine source code authorship. *White paper: department of computer science, university of Wisconsin.* 2000.

[4] D. Zou, W. Long, and Z. Ling. A cluster-based plagiarism detection method - Lab report for PAN at CLEF *In Proceedings of the 4th Workshop on Uncovering Plagiarism, Authorship, and Social Software Misuse* 2010.

[5] J. Son, S. Park, and S. Park. Program Plagiarism Detection Using Parse Tree Kernels. *PRICAI'06 Proceedings of the 9th Pacific Rim international conference on artificial intelligence:* 2006. pp 1000–1004.

[6] C. Liu, C. Chen, J. Han and P. S. Yu. GPLAG: Detection of Software Plagiarism by Program Dependence Graph Analysis. *In Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*: 2006. pp 872-881.

[7] S. Harihan. Automatic Plagiarism Detection Using Similarity Analysis. *The International Arab Journal of Information Technology:* Vol. 9, issue 4. 2012.

[8] Z. Duric, and D. Gasevic. A Source Code Similarity System for Plagiarism Detection. *The Computer Journal*: Vol. 56, issue 1, 2012. pp 70-86.

[9] L. Zhang, D. Liu, Y. Li, and M. Zhong. AST-Based Plagiarism Detection Method. *In: Wang Y., Zhang X. (eds) Internet of Things. Communications in Computer and Information Science.* Springer, Berlin, Heidelberg, Vol. 312. 2012.

[10] D. Ganguly, G. J. F. Jones, A. Ramı´rez-de-la-Cruz, G. Ramı ´rez-de-la-Rosa, and E. Villatoro-Tello. Retrieving and classifying instances of source code plagiarism: *Information Retrieval journal*: 2017. pp 1-23.

[11] E. Flores, A. Barro ´n-Ceden ˜o, P. Rosso, and L. Moreno. Towards the detection of cross-language source code reuse. In *Proceedings of the 16th international conference on applications of natural language to information systems*: 2011. pp. 250–253.

[12] S. Narayanan and S. Simi. Source Code Plagiarism Detection and Performance Analysis Using Fingerprint Based Distance Measure Method. *In Proceedings of 7th International Conference on Computer Science Education ICCSE '12. IEEE*: pp 1065-1068.

[13] R. Marinescu. Accessing Technical Debt by Identifying Design Flaws in Software Systems. *IBM Journal of Research and Development:* Vol. 56(5), 2012. pp 1-9.

[14] S. Ion and I. Bogdan. Source Code Plagiarism Detection Method Using Protégé Built Ontologies: *Informatics EconomicsJournal:* Vol. 17, 2013. pp 75-86.

[15] T. Ohmann and I. Rahal. Efficient clustering-based source code plagiarism detection using PIY. Journal of *Knowledge and Information Systems*: vol. 43, 2014. pp 445-447.

[16] J. Zhao, K. Xia, Y. Fu, and B. Cui. An AST-Based Code Plagiarism Detection Algorithm: *10th International Conference on Broadband and Wireless Computing, Communication and Application*. 2015.

[17] N. More, A. A. Bhootra and C. A. Patel. Plagiarism Detection in Source Code. *IJIRST –International Journal for Innovative Research in Science & Technology*: Volume 1, Issue 10 | March 2015 ISSN (online): 2349-6010, 2015. pp 109-112.

[18] N. Shah, S. Modha, and d. Dave. Differential Weight Based Hybrid Approach to Detect Software Plagiarism. *In Proceedings of International Conference on ICT for Sustainable Development: Vol. 409,* 2016. pp 645-653.

[19] O. Karnalim. Detecting Source Code Plagiarism on Introductory Programming Course Assignments Using a Bytecode Approach. *The 10th International Conference on Information, Communication Technology and System (ICTS), Surabaya, Indonesia:* IEEE, 2016. pp 63-68.

[20] O. Karnalim. A Low-Level Structure-based Approach for Detecting Source Code Plagiarism. *IAENG International Journal of Computer Science*: volume 44, 2017. pp 4.

[21] M. Duracik, E. Kirsak, and P. Hrkut. Source Code Representations for Plagiarism Detection. *Springer International Publishing AG, part of Springer Nature:* CCIS 870, 2018. pp 61–69.

[22] M. Duracik, E. Kirsak, and P. Hrkut. Scalable Source Code Plagiarism Detection Using Source Code Vectors Clustering. *IEEE Journal:* 2018. pp 7-18

[23] O. Karnalim. Source Code Plagiarism Detection in Academia with Information Retrieval: Dataset and the Observation. *Jornal of Informatics in Education*: Vol 18(2), 2019. pp 321-344.

[24] Dr. R, Kulkarni1 and K., Apana. A Novel Approach to Restructure the Input Java Program. *Journal of advanced networking and applications*: Volume: 12 Issue: 04 Pages: 4621-4626(2021).

[25] R. Kulkarni and P., Pani. Abstraction of UML Class Diagram from the Input Java Program. *Journal of advanced networking and applications*: Volume: 12 Issue: 04 Pages: 4644-4649(2021).