Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Plagiarism Detection in Verilog Hardware Description Language Programs using Machine Learning

Anusha Gupta

MSc. in Computer Science - Data Science
in completion of the Postgraduate Taught Course
School of Computer Science & Statistics
Trinity College Dublin, Ireland

Supervisor: Prof. John Waldron

September 2020

Declaration

I declare that this thesis has not been submitted as an exercise for a degree at this or any other university and it is entirely my own work.

I agree to deposit this thesis in the University's open access institutional repository or allow the library to do so on my behalf, subject to Irish Copyright Legislation and Trinity College Library conditions of use and acknowledgement.

Anusha Gupta

September 6, 2020

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Anusha Gupta

September 6, 2020

Acknowledgments

On the completion of this dissertation, I would like to extend my sincere and heartfelt obligation towards all the people who have helped me in this endeavour.

I am extremely thankful to my supervisor, **Prof. John Waldron**, for his continuous guidance and encouragement to accomplish this dissertation.

I also acknowledge with the deep sense of reverence, my gratitude towards my parents and sister, who have always supported me through out the course. At last but not the least, gratitude goes to my friends who directly or indirectly helped me to complete this dissertation.

Anusha Gupta

September 6, 2020

Plagiarism Detection in Verilog Hardware Description Language Programs using Machine Learning

Anusha Gupta, Master of Science in Computer Science - Data Science
University of Dublin, Trinity College, 2020

Supervisor - Prof. John Waldron

In this dissertation, we study how we can apply machine learning algorithms to detect plagiarism in Verilog Hardware Description Language programs. It presents a novel technique which can identify similarities between VHDL programs and gives the most similar files among all the files trained. The second part of the research focuses on evaluating the trained machine learning models and identifying which is the most accurate among those in providing the results of plagiarized files. The training as well as testing datasets used in this study consists of the code metrics which are annotated in the form of tokens representing the structure of each VHDL program. Several machine learning algorithms were compared in this research to identify the best performing algorithm, among which Stochastic Gradient Descent classifier came out to be the most accurate with 82.3% accuracy. Lastly, the results of this algorithm were used to detect the similar files and identify plagiarism within them.

Contents

List of Figures

List of Tables

List of Abbreviations

- **VHDL** - Verilog Hardware Description Language
- **AUC** - Area Under Curve
- **PDG** - Program Dependence Graph
- **GCC** - GNU Compile Collection
- **RTL** - Register Transfer Language
- **SVD** - Singular Value Decomposition
- **NLP** - Natural Language Processing
- **DTM** - Document Term Matrix
- **LDA** - Latent Dirichlet Allocation
- **RKR** - Running KARP Rabin
- **GST** - Greedy String Tiling
- **AST** - Abstract Syntactic Tree
- **CV** - Cross Validation
- **KNN** - K Nearest Neighbour
- **SGDC** - Stochastic Gradient Descent Classifier
- **MLPC** - Multi-Layer Perceptron Classifier
- **GD** - Gradient Descent
- **SVM** - Support Vector Machine
- **LDA** - Linear Discriminant Analysis

- **TF-IDF** - Term Frequency-Invert Document Frequency
- **TP** - True Positives
- **TN** - True Negatives
- **FP** - False Positives
- **FN** - False Negatives

Chapter 1

Introduction

This dissertation explores the area of plagiarism detection in source codes. The main focus of this research is on Verilog Hardware Description Language programming assignments submitted by students of Trinity College Dublin. The primary aim is to propose a machine learning technique to detect plagiarism in VHDL programs. Also, it focuses on obtaining the most accurate machine learning algorithm among the six compared which can help in detecting the plagiarized files more accurately.

1.1 Motivation

Plagiarism and cheating are a major problem in academia since decades. Many universities have taken measures to address this issue in their own way by adopting various rules and regulations, providing general education programs for students as well as purchasing plagiarism detection softwares. J. Zobel [47] has mentioned one such case study of plagiarism which occurred in the school of computer science Information Technology at RMIT University. This incident pointed out the discovery of an external private tutor who was selling assignments to the students. According to J. Zobel [47], he was active for at least three semesters and was not only selling solutions but was also taking exams on particular students' behalf. However, the case was out in media and as plagiarism was considered as a serious offence by the community, the 'private tutor' and students were sentenced in court. Moreover, through his study, he explained that a large number of invigilation processes are required if students are to be convinced to do their own work, that is, plagiarism is encouraged by lazy invigilation and the universities who are avoiding the issue of plagiarism are undermining the quality of their degrees. Therefore, systematic invigilation and cross checking is needed to eliminate plagiarism from academia.

This dissertation is a research conducted so that the problem of plagiarism faced by instructors in programming assignments can be solved. Although there are various researches done and tools

developed to detect plagiarism in different programming languages, VHDL programs were not the focus of many of those tools and researches. Hence, this research focuses on the implementation of plagiarism detection technique for VHDL programs which were submitted by students as their assignments in a course module. Not only plagiarism detection in student's assignments, the proposed technique can also be implemented in wide variety of areas mentioned as follows [29]:

- **Criminal Prosecution** – Identification of author of a malware belonging to suspects.
- **Corporate Litigation** – Identification of author of codes in case of violation of a no-compete clause of contract by an employee.
- **Plagiarism Detection** – Identification of author in case of cheating in academia.

The plagiarized program can be defined as the programs which have been copied from other programs by making small transformations. There are several scenarios in which plagiarism can be occurred such as [22] [34]:

- Willingness of students to share the code with other students in case the assignment deadline is approaching.
- Students working together on an assignment can result in submission of similar programs with naming and formatting differences.
- Assignments can be stolen through shared printers and workstations.
- Programs from previous semesters can be used as only specifications are modified from one semester to next.
- Small assignments, not plagiarized though, seem to be similar because several students arrive at the same design independently.

There are various transformations made by students in their assignments to avoid being caught. Those transformations in source codes can take various forms and some of them are as follows [31]:

- **Format Alteration** – This is simply the insertion and removal of blank spaces and comments.
- **Identifier Renaming** – Altering identifiers' names is another common practice for plagiarism as it does not violate program's correctness.
- **Statement Reordering** – The statements which do not have sequential dependencies such as declarations can be reordered easily.

- **Control Replacement** – There are many replacements for simple codes in programming languages such as while loops can replace for loops, the conditions of if else can be interchanged for the same logic, etc.
- **Code Insertion** – Codes which do not alter the original logic of the program can be inserted to disguise plagiarism.

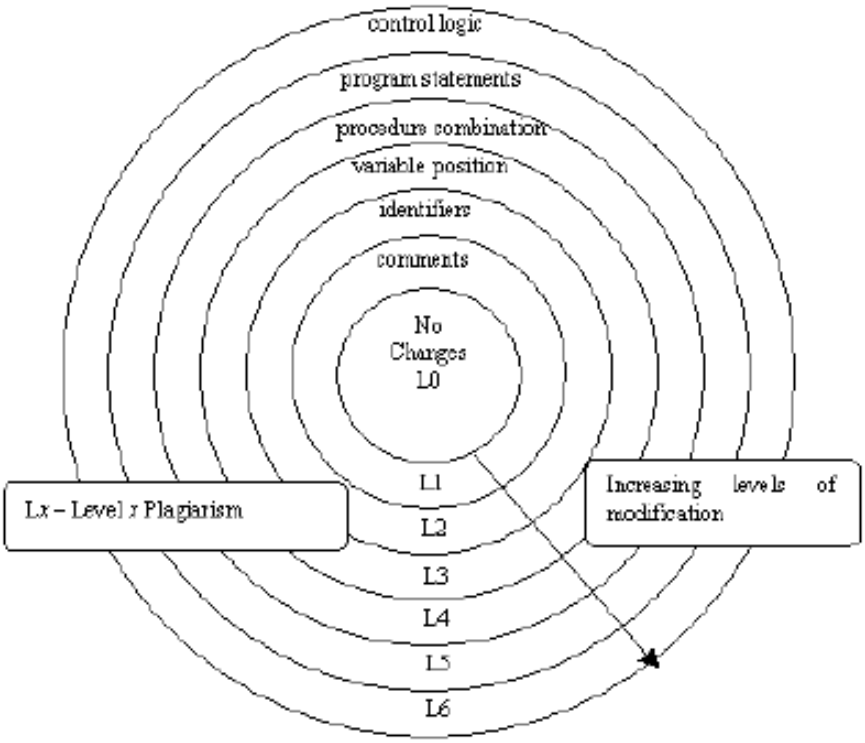

Figure 1.1: Program Plagiarism Spectrum [11]

Some of the above mentioned types of plagiarism can be detected by instructors themselves without the help of any tool. However, minute differences would need a tool or a technique to detect plagiarized files. Three of the major techniques implemented in plagiraism detection tools that are mentioned by Chao Li [31]:

- **String Based** – The statements in the programs are considered as strings and are compared to find the same strings. However, blanks, comments and identifier renaming are ignored in this method which makes it fragile.
- **Abstract Syntax Tree based** – The program is parsed into an abstract syntax tree as shown in Figure 1.2 and it is searched for similar subtrees which is labelled as plagiarized if found.
- **Token based** – In this method, identifiers and keywords are used as token sequences and

duplicate sequences are considered as plagiarism. However, it is fragile to reordering and code insertion. JPlag and Moss are some of the tools using this method.

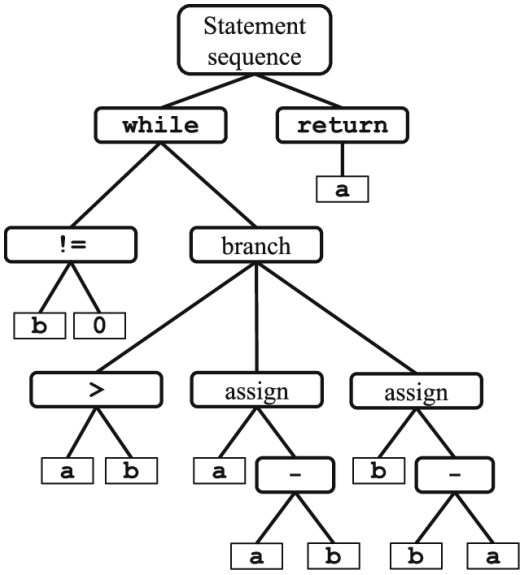

Figure 1.2: Abstract Syntax Tree [42]

Figure 1.1 shows a diagram of six levels of program modification in plagiarism proposed by Faidhi and Robinson [12]. According to this diagram, Level 0 is the lowest level of plagiarism which consists of copied programs without modifications whereas level 6 is the highest level which consists of programs with modified control logic. It is difficult to detect plagiarism on the basis of structural properties when the level is 4 or above. The attribute counting techniques, however, do not rely on this diagram and are not suffering from the problem mentioned above [5]. Hence, this dissertation discusses machine learning method based on attribute counting technique, in other words, extraction of code metrics.

1.1.1 Verilog Hardware Description Language

Verilog Hardware Description Language is a text based description of digital logic circuits and their behaviour over time. It has syntactical similarities to software programming languages but are fundamentally different as it helps in describing the time and signal strengths [40]. It is basically used for simulation and testing of a design of an actual circuit and is not very sensitive to statements ordering like other procedural languages.

In this research, the data consists of VHDL assignments of 18 students undertaking a four years honour degree in computer science at Trinity College Dublin. The assignments are a part of Digital Logic Design course studied by students for two semesters. The VHDL is taught in the second semester which is the main focus of this dissertation. These assignments include structure verilog for

simple combinational logic problems to difficult sequential logic problems using shift registers, counters, etc [40] [33]. The Verilog compiler and simulator used by the course is the Icarus Verilog tool which can be used on multiple operating systems. Although this information is useful for context, further research in this dissertation does not require any understanding of VHDL.

1.1.2 Machine Learning

Machine learning, as the name suggests, is an application of artificial intelligence that enables machines to automatically learn and improve from experiences by accessing data. It helps in building models to analyze complex data and provide accurate results. Some of the things needed to create good machine learning systems are data preparations capabilities, scalability, algorithms and ensemble modeling. Machine learning can be categorized into three categories:

- **Supervised Learning** - In this type, the data contains a target variable according to which predictors learn the outcome and predict or classify the testing data as per the specifications of the problem. The labeled dataset can be used to evaluate the accuracy of the generated model. For Example, Linear Regression, Support Vector Machine, etc.
- **Unsupervised Learning** - This is used when the dataset is neither classified nor labeled. It helps in exploring the data and describing the hidden structure of the unlabeled dataset. For Example, Clustering, etc.
- **Reinforcement Learning** - This learning method produces actions by interacting with the environment and is penalized or awarded according to the action chosen. It learns by using trial and error method and determines the ideal behaviour in order to maximize its performance. The reward feedback is provided for the agent to learn the best possible action. For Example, Deep Neural Network, etc.

In this research, I am using supervised learning algorithms to determine the classification of the source code files according to their programmers. The training dataset used is the labeled one and consists of programmer as its target variable. All other variables are the code metrics extracted from the programs written by students which provide us with the information about the characterstics of those programs. The code metrics generated from the programs are line length, line words, indent space, inline space, number of underscores, number of spaces, number of lines, identifiers with underscores, number of identifiers and number of keywords. The training data contains the assignments of 18 students with 15 assignments per student and testing data contains 63 anonymous assignments without target variable, that is, Programmers. Furthermore, the algorithms used for classifying and

detecting plagiarism are K-Nearest Neighbours, Decision Tree, Random Forest, Stochastic Gradient Descent, Linear Discriminant Analysis and Multi-layer Perceptron.

1.2 Objectives

The primary objective of the dissertation is to train a machine learning algorithm that is capable of detecting similar programs which provide us with the information about the plagiarism between those programs. The primary motivation behind it was to develop a machine learning model that can help in detecting plagiarism in VHDL programs as there are very limited researches done in this area. The secondary objectives that were a part of this research are:

1. Preparation of code metrics dataset for training of machine learning algorithms. It consists of tokens representing the structure of VHDL programs being trained.
2. Exploration of machine learning algorithms, that is, which will be the best for classification of similar programs and detecting plagiarism.
3. Exploration of the measurements to evaluate the quality of machine learning algorithms used for training the data and identifying the most accurate among them.

The primary programming language used for writing the source codes for collection of data as well as writing program that would fit and classify that data is Python Programming Language. As the starting point, the existing algorithm was implemented as mentioned in [5]. However, the code was re-implemented as per the requirements of VHDL programs as the existing code was developed for JAVA programs. Also, the algorithms used in this research are different from the existing research and according to new inclusions in Scikit-Learn.

1.3 Overview of the Dissertation

The layout of the dissertation is as follows:

Chapter 2 Literature Review mentions some of the current researches in this field. First few sections such as GPlag, XPlag, Fuzzy-Based Approach consist of various techniques introduced using different algorithms in this area. Then, it introduces some of the current trends being used in plagiarism detection such as Karp Rabin technique, N-Grams, etc. The last section gives the insight about some of the researches done using machine learning.

Chapter 3 Methodology is the implementation of the research in detail. It covers four main areas:

- Machine Learning Algorithms used in this research.
- Dataset generation for training and testing of the algorithms.
- Implementation architecture of the technique.
- Evaluation techniques used for evaluating the performances of the machine learning algorithms as well as detecting similar files to detect plagiarism.

Chapter 4 Results and Discussion presents the results of two researches done in this study. Firstly, the results of best performing machine learning algorithm for plagiarism detection using Accuracy and AUC value and secondly, the results of the plagiarized files using confusion matrix as evaluation criteria.

Chapter 5 Conclusion and Future Work provides the main outcome of this research and proposes ways to improve the results in the future.

Chapter 2

Literature Review

Sections 2.1 and 2.2 explain some of the researches done in plagiarism detection using various techniques other than machine learning. Section 2.3 mentions some techniques in which machine learning was used and how the code metrics dataset generation is important for it.

2.1 Introduction

Plagiarism is a major problem in academia since decades and to tackle this problem, researchers have tried to develop various plagiarism detection tools for different programming languages including Java, C, fortran, HDL, etc. Some of these researches that motivated me in taking up this research problem include:

2.1.1 GPlag

Chao Li [31] introduced a new technique for plagiarism detection in programming languages named GPlag which uses Program Dependency Graphs (PDGs). A PDG as shown in Figure 2.1, is a graph representation of the source code of a procedure, where statements are represented by vertices and data and control dependencies between statements by edges [31]. The main aim of using PDGs in this research was that these graphs are almost unchanged to free modification of the code by the plagiarist and to change the graphs, a much higher effort is needed to disguise the original code which contradicts the purpose of plagiarism. GPlag [31] is an implementation of two major techniques that helped in increasing the robustness, efficiency and effectiveness of the tool. Firstly, Subgraph Isomorphism which helps in identifying different types of disguises such as format alteration, code insertion, identifier renaming, etc. and secondly, Lossy Filter which helps in pruning of the trees of plagiarized part using G-test.

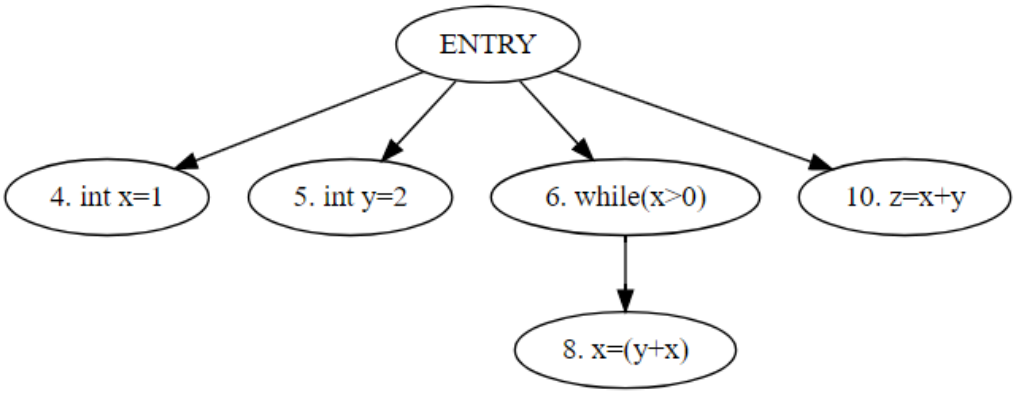

Figure 2.1: Program Dependence Graph [9]

2.1.2 XPlag

In another research, C. Arwin [3] proposed a novel approach, XPlag, to detect plagiarism which not only works with single programming language but it also detects inter-lingual plagiarism, that is, plagiarism between source codes that are converted from one programming language to another. According to this research, there are two ways to address inter-lingual plagiarism. Firstly, by comparing token produced by existing plagiarism detection approaches that support multiple programming languages and second is by comparing intermediate code produced by a compiler suite that supports multiple languages. However, they used the latter approach as former required different scanner and parser for each language. As a compiler suite, they used GNU Compile Collection (GCC) which supports multiple programming languages such as C, C++, Java, Fortran and Objective C. XPlag [3] works in two phases: Indexing in which intermediate code, Register Transfer Language (RTL), generated from source code files are indexed and Detection in which the indexes are searched using RTL and the programs are ranked in the order of decreasing similarity, calculated by Okapi BM25 similarity function.

2.1.3 Fuzzy-Based Approach

This research paper [1] introduced a new Fuzzy-based approach for plagiarism detection in programming languages using Fuzzy C-Means and the Adaptive-Neuro Fuzzy Inference System. According to G. Acampora [1], fuzzy clustering is a suitable approach for source code plagiarism detection as it has the capability to capture qualitative and semantic elements of similarity. One of the major advantages of using fuzzy based approach is that it does not require a parser or compiler for each programming language to detect plagiarism unlike string matching algorithms based on structural characteristics. This novel approach, first, uses singular value decomposition to remove noises from the source code

files, then uses Fuzzy C-means clustering to cluster files based on similarity and finally optimize the performance using Adaptive Neuro-Fuzzy Inference System. The pre-processing of source code is required to reduce the data size and capture the semantic representation of each source code more effectively. It includes conversion of upper case to lower case, removing numeric characters, semi-colons, colons, single letters, etc. A term by file matrix is produced as a final output of pre-processing which contains the frequency of each dictionary term that appears in a source code file and then normalizes the frequencies using normal global weighting function and document length normalization. Further, Singular Value Decomposition (SVD) is used to reduce dimensional space of data. The final matrix is fed into the learning algorithm which is divided into two steps:

- **Fuzzy C-Means** – This generates a collection of clusters, each containing similar source code files.
- **ANFIS** – This optimizes the training data and the performance is measured using the root mean square training errors.

The final performance evaluation is done through Recall, Precision, Fscore, Specificity, Accuration and Pearson Correlation Coefficient.

2.1.4 Gene Sequence Inspired Approach

In this research, Mark C. Johnson [22] proposed a technique for plagiarism detection in HDL programs which is similar to evaluating similarity and ancestry of gene sequences. This approach of plagiarism detection is an application of compression measurements of two versions of VHDL source code: partially tokenized and literal [22]. In this technique, the files are compressed using gzip and for each pair of students, sizes of the concatenated compressed files are compared with the individual compressed files and similarity metrics are computed. This is repeated for both the versions of VHDL codes. Once the similarity analysis is complete, the instructors have to inspect the most suspicious pairs of source code to determine where plagiarism has occurred [22].

2.1.5 Parse Tree Similarities

In a research, C.W. Anderson [11] presented a source code plagiarism detection method using parse tree. The parse tree, Figure 2.2, is a graph that is built from the lexical of a program and exhibits the structure for it. Compilers use it to guide compilation of the programs. The algorithm proposed by C.W. Anderson [11] for detecting plagiarism using parse tree was: Firstly, parse each program. Then, convert each parse tree into a string and lastly, calculate the similarity between programs using Greedy String Tiling method. Some of the limitations of this technique include coupling of unrelated parts of

the program and it only supports Java language and has not been tested for other programming languages.

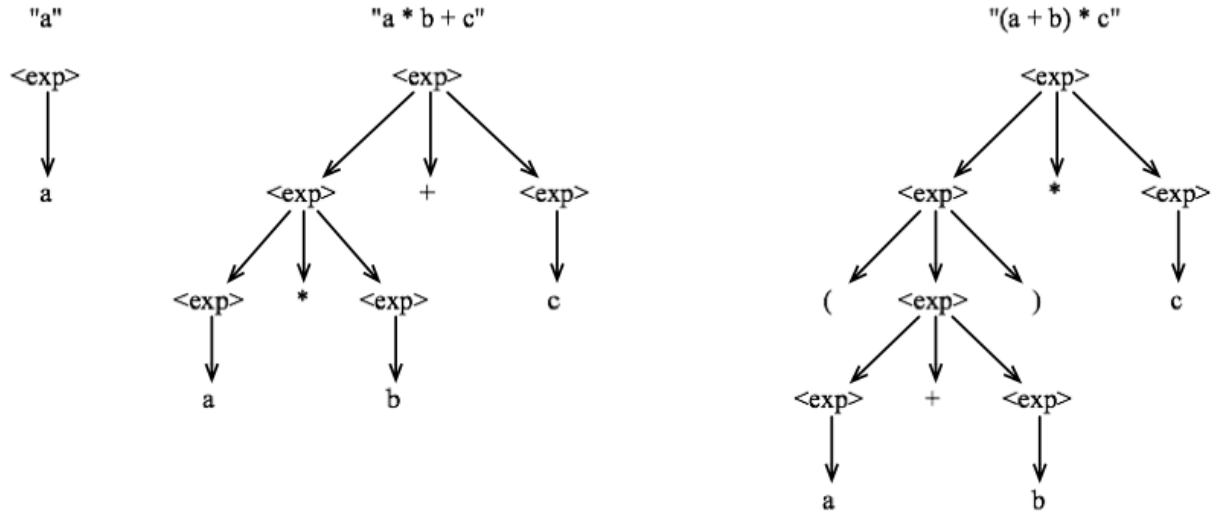

Figure 2.2: Parse Tree [19]

2.1.6 Algorithm-Based Approach

Ottenstein [35] in his research discussed a quantification algorithm through which equivalent programs are given equal values. This can be represented as a function, f, imposing a partitioning on the set of programs such that if X and Y are unique source codes with f(X)=f(Y), then one of X and Y is a plagiarized version of the other [35]. However, this function is difficult to obtain as identical programs can possibly be written independently as well as plagiarism is quite different from paraphrasing. Therefore, Ottenstein [35] tried to find an approximation that could map similar programs into same partitions. For this, he used four parameters: n1(number of unique operators), n2 (number of unique operands), N1 (total number of occurrences of operators) and N2 (total number of occurrences of operands). He used a multivariate normal density function, g, which is determined by the means vector and the covariance matrix. The closeness of the value of function, g, to the means vector indicates the accuracy of plagiarism of each partition.

Another study by Alan Parker [36] presented a computer algorithm based on string comparisons used for the detection of plagiarism in source codes. Its implementation includes removal of comments, blank spaces and extra lines, comparison of character strings in two files using UNIX utilities - grep (search strings), wc (count words and lines) and diff (compare two files), and counting the similarity percentage of characters called character correlation. According to Alan Parker [36], it is an

easy implementation, however, it requires a substantial amount of computer time.

One of the researches done by professors of my university [40], Trinity College Dublin, is a contribution in plagiarism detection in Verilog programs. They examined and implemented three algorithms to calculate similarity between 11 types of student's Verilog programs. For each assignment submitted by the students, they prepared three versions using flex and C code: original submission, cleaned (comments amd white spaces deleted) and tokenised (identifiers and numeric literals replaced by constants). Three measures of similarity which they used were [40]:

- **Jaccard measure** - For two files X and Y, the formula is J(X, Y) = |X and Y|/|X or Y|
- **Tversky measure** - For two files X and Y, the formula is T(X, Y) = |X and Y| / |X|
- **Sequence measure** - For two files X and Y, the formula is R(X, Y) = 2|M| / |X| + |Y|, where M is the number of matched elements.

To show the distribution of similarities, they plotted violin plots with median and inter-quartile range and found similar results for each algorithm. However, one of the dangers with this research is the over processing as all possible differences might be eliminated and the programs are made appear similar [40].

2.2 Current Trends in Plagiarism Detection

Ducarik in his study [10] analysed the state of the art in source code analysis for plagiarism detection. According to him, source code analysis is divided into three levels: as a plain text (explores the meaning of the text), token based (does not explore the meaning of the text) and as a model. Natural Language Processing (NLP) is one of the famous methods which process natural language using computers. Using NLP, the terms can be extracted out of the source codes in the form of tokens and then analysed using various classification methods. Another approach used when several source codes are involved is Document-term Matrix (DTM). The matrix lines are a vector for each source code and represent the source codes as rows and the terms as columns i.e. describes the occurrence of terms in each source code. However, the main problem with DTM is its lower memory capacity which can be solved using sparse matrices but it adds up to computational complexity. Third approach discussed by Duracik is Latent Dirichlet Allocation (LDA) which is used to transform the source code into topic models i.e. n of important terms. JPlag and MOSS are some of the techniques which use this approach for plagiarism detection. Their initial stages are code-purification and tokenization (converting source code into individual sequence of characters) and post processing can be done through various methods such as:

- **N-grams** – In this method, documents are divided into list of substrings of length n with their frequency.
- **Winnowing** – It uses hashing to select a subset of n-grams of a document as its imprint. This method is used by MOSS.
- **Running-KARP-Rabin Greedy-String-Tiling (RKR-GST)** – It has two phases: RKR which is an algorithm for substring matching and used to find common substrings and GST which is used to find more precise substrings. It is used by JPlag.

The third method mentioned by Ducarik [10] is source code model analysis and one of the applications of this method is Abstract Syntactic Tree (AST). This tree is formed using the tokens stream of source code which represents the logical structure without pointless parts. Hashing methods are used for comparison of ASTs as it works relatively quickly.

Apart from this, some of the existing successful plagiarism detection tools are as follows:

1. **SIM** - According to C. Arwin [3], this tool works with parameters like correctness, style and uniqueness. The tokens are generated by parsing each program using flex lexical analyser and then compared based on scores generated by comparing token strings of two programs. It can handle name changes and reordering of statements and functions.
2. **YAP3** -It detects plagiarism in two phases. First, token sequences are generated and mapped to their base equivalents. Second, the matches are obtained using Karp-Rabin greedy string-tiling algorithm.
3. **JPlag** – This is a web service and uses the same comparison algorithm as YAP3. However, to make the tool more accurate, it includes some context of the program structure into the token strings. For example, it uses *BeginMethod* token to indicate open braces at the start of the method and *OpenBrace* for any other open braces [41].

2.3 Machine Learning in Plagiarism Detection

Machine learning, as the name suggests, is an application of artificial intelligence that enables machines to automatically learn and improve from experiences by accessing data. It powers multiple applications such as recommendation systems, search engines, voice assistants, social media, etc. Many researchers used machine learning in their researches for plagiarism detection as well. Hence, this was the major reason of testing plagiarism using Machine Learning in this dissertation. N-grams, tf-idf and code metrics are some of the current trends used for developing machine learning techniques

for source code plagiarism detection. One such research was done by D. Heres [17] in which he developed a machine learning tool, Infinite Monkey, to improve plagiarism detection in programming languages. This system could identify similarities between programming files using two methods: N-grams, tf-idf weighting and cosine similarity were used for fast retrieval of similarities in programs and neural network model was used for classification of large source code plagiarism dataset. For pre-processing of the codes, he used an obfuscation tool which reordered import statements, class members, removed methods, white spaces and new lines and renamed identifiers. However, according to this study [17], the trained model did not generalise well to the source code plagiarism tasks they evaluated.

2.3.1 Code Metrics

While studying this approach, another thing which I came across is the importance of metrics selection for training the model using machine learning, as used in this dissertation. R. Lange [29] proposed a method of metric extraction out of source codes to perform author identification. Their study involves representation of 18 metrics in the form of histogram distributions. This research paper [29] has two goals, firstly, how well can histogram distribution of code metrics identify the style of a developer and secondly, what is the best combination of metrics for identification of authors. The histograms were generated by plotting metric vs its frequency graph for each document and normalized it by dividing the frequencies by sum of all frequencies. For author identification, nearest neighbour classification algorithm was used as it can produce a ranked list of developers in order of descending likelihood of authorship. The best combination of code metrics, according to this study, came out to be inline space, inline tab, line length, line words, period, trail space, trail tab and underscore. However, they concluded that this set of metrics may change according to the dataset as well as if one set of combination of metrics identifies the author of one set of code, the same combination will not necessarily identify the author of another set of code. Similarly, Faidhi [12] in one of his researches provided an empirical program analysis experiment that determines the programming measures that might aid in detection of plagiarism. To implement this approach, Faidhi [12] simply collected data from source codes that can study several features of a program. He selected two sets of measures: the set which reflects features of program sets such as number of characters per line, comments, indented lines, blank lines, variety of identifiers, etc. and the set which tells about the intrinsic and hidden features of a program's structure such as metrics of flow of control diagrams. Then performed tests by deleting one measure at a time from the set, correlated the remaining set of measures between each student, similarity indicator was calculated using a similarity gauge and plagiarized files were counted. With this experiment, he proved that empirical approach can detect plagiarism successfully and is sensitive

enough to avoid misleading an evaluator [12].

2.3.2 Scikit-Learn

As explained by F. Pedregosa [38], Scikit-Learn is a module written in Python Programming language that includes wide variety of state of the art algorithms of machine learning for unsupervised and supervised problems. As python has a high-level interactive nature as well as ecosystem of scientific libraries, it is considered as the perfect choice for algorithmic development and exploratory data analysis [38] [46]. Scikit-Learn has many advantages over other machine learning modules such as it is totally dependent on numpy and scipy for easy distribution and imperative programming remains its focus. Moreover, some of the major technologies of this module are:

1. **Numpy** - It is used for arithmetic operations and data and model parameters. The input data is converted to numpy arrays to integrate it with other python libraries.
2. **Scipy** - It is an algorithm used for statistical functions, matrix representation and linear algebra.
3. **cython** - It is a language that combines C with Python and helps in easily reaching the performance of compiled languages.

Apart from these libraries, Scikit-Learn also facilitates estimators that consists of fit and predict functions and some can also be used to transform data. Furthermore, it can help in the evaluation of performance of these estimators and selection of parameters using GridSearchCV.

Besides using machine learning for plagiarism detection, this dissertation also focuses on comparison of performance of algorithms in Scikit-Learn module for this particular problem. For its implementation, my major motivation came from a research done by U. Bandara [5] where he compared three machine learning algorithms for detecting source code plagiarism in Java programming language. The algorithms selected by him were Naive Bayes classifier, KNN classifier and Adaboost algorithm. To compare these, he calculated the accuracies of all the algorithms and computed the confusion matrices to check the total failures and success rate for each algorithm. According to this research, the best performing algorithm for plagiarism detection came out to be Adaboost algorithm. However, I will be comparing six machine learning algorithms in this dissertation by not only computing the confusion matrices but also comparing the values of Precision, Recall, Fscore and AUC values of each algorithm.

Chapter 3

Methodology

Section 3.1 explains all the machine learning algorithms used in this research for comparison as well as plagiarism detection. Section 3.2 explains how the dataset is generated and what code metrics are used in it. Section 3.3 mentions all the evaluation metrics used for evaluating the performance of the algorithms.

3.1 Algorithms for Source Code Plagiarism Detection

The periods have changed vastly over the past few decades. The computing moved from mainframes to computers to cloud. With this evolution, various tools and techniques have also been developed to boost computing in which Machine Learning algorithms are one of the main contributors in this period. There are three categories in which these algorithms are divided:

- **Supervised Learning** - The dataset used in this type consists of a target variable along with other predictor variables. A function is generated by training the data that maps predictors to desired targets until the highest accuracy is achieved. Examples of supervised learning are KNN, Random Forest, Linear Regression, Naive Bayes, etc.
- **Unsupervised Learning** - In this type of algorithm, dataset does not contain target variable. Instead, it clusters the data in different groups. Examples are K-means, etc.
- **Reinforcement Learning** - This type of algorithm helps machine to make the decision on its own by training and learning from the experience and capture the best possible knowledge. Examples are Markov Decision Process.

In this dissertation, I have used Supervised Learning classification algorithms as the dataset consisted of a target variable as well as it was the best suited classification algorithms for source code

plagiarism detection problem. The algorithms which I implemented and compared in this dissertation are:

3.1.1 K-Nearest Neighbour Classifier

The K-Nearest Neighbour (KNN) is a simple yet effective classification algorithm of machine learning [16]. It stores all cases on the basis of similarity measure, that is, by calculating distance between test data and training data, Figure 3.1. The working of the algorithm is as follows:

1. Let t be a data which has to be loaded for training and testing.
2. Choose the value of K, nearest data points, which could be a random value.
3. For every value in the test data, distance is calculated from the training data points using any one of the three functions: Euclidean distance, Manhattan distance or Hamming Distance. However, euclidean distance is the most commonly used function.

$$dist(A,B) = \sqrt{\Sigma_{i=2}^{m}(x_i - y_i)^2}$$

4. Then, points are sorted in ascending order based on the distance values.
5. Top K rows are selected from the sorted list and finally, the most frequent class is assigned to the test value.

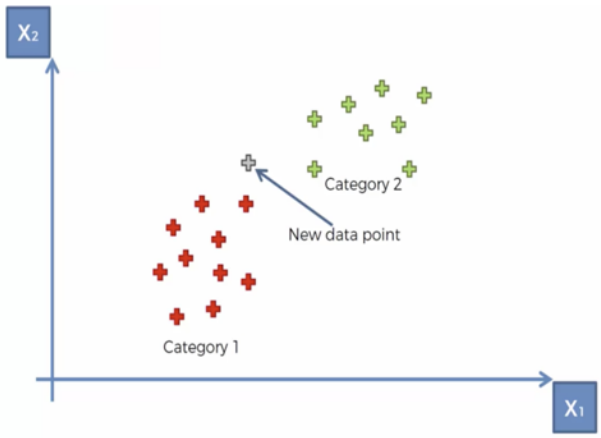

Figure 3.1: Graphical Representation of K-Nearest Neighbour [18]

Selecting the value of K is the most important part of this algorithm as success of classification depends on it. The simplest way of choosing the value of K is by running the algorithm multiple times with different values of K and then, selecting the best performing one. In this research, the default

value of K, that is, 5 was the best performing one. However, there is a drawback with KNN algorithm, that is, all the computation take place at classification time rather than when the training examples are first encountered which makes its classification cost high [16].

3.1.2 Decision Tree Classifier

Decision Tree is a flow chart like tree structure in which each internal node denotes a test on an attribute, each branch represents the outcome of test and every leaf is the class label [37]. It consists of two categories:

- **Classification Trees** - In this, target variable consists of discrete values.
- **Regression Trees** - In this, target variable contains continuous values.

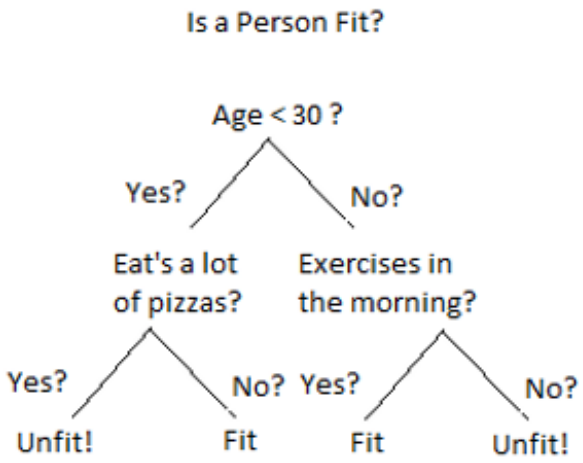

Figure 3.2: Decision Tree [45]

The algorithm for Decision Tree is as follows:

1. The best attribute of the dataset is considered to be the root node.
2. Training data is divided into subset containing similar values for an attribute.
3. Repeating above steps for every subset until we get a branches of trees with leaf nodes.
4. After the tree is generated, firstly, the test value is compared to the root value and the next node is selected on the basis of the comparison.
5. It keeps on comparing the test value with every internal nodes until it reaches a leaf node with predicted class.

It follows Sum of Product representation which is also known as Disjunctive Normal Form, that is, from root to leaf node with same class is calculated using product and the different branches that ends in that class is formed using addition. One of the main challenges in decision tree algorithm is selecting the root node which is done by using either of the two criteria:

- **Information Gain** - For this, variables are considered categorical and entropy measure is calculated using which information gain can be calculated.

$$H(X) = E_X[I(x)] = -\Sigma p(x) log p(x)$$

- **Gini Index** - For this, variables are considered continuous and it measures the occurrence of incorrectly chosen element so the lower index is preferred.

Some of the many advantages of using Decision Tree algorithm include its implementation does not require any domain knowledge , it can handle hidimensional data, is simple, fast and understandable and have good accuracy [37].

3.1.3 Random Forest Classifier

Random Forest is an ensemble classification algorithm in machine learning. It generates multiple decision trees and applies majority voting to combine the outcomes of these trees [7] [27]. The randomization in this algorithm is done is two ways: sampling of data is done randomly for bootstrap samples and input variables are selected randomly for generation of individually decision tree [7] [27]. Decision trees in this algorithm can be generated by following steps:

1. The records are randomly sampled with replacement from the original data which is called bootstrap sampling. This is the training set used to generate trees.

2. From this data, *m* variables are selected randomly which is kept constant until tree is grown fully and the best split is used to the split the node.

3. The number of trees generated and the depth of the trees can be controlled using parameters Ntree and node size, respectively.

Once the forest of decision trees is generated, test data is ran through all of them giving the classification for it. The results from each tree are combined and the classification with majority votes is assigned to the test data. At the time of bootstrap sampling, some data is left out which is known as out-of-bag data and is used to estimate error of individual tree. The formula for this is as follows:

$$PE^* = P_{x,y}(mg(X,Y)) < 0$$

Figure 3.3: Representation of Random Forest Algorithm [43]

The formula for margin function is as follows:

$$mg(X,Y) = av_k I(h_k(X) = Y) - max_{j \neq Y} av_k I(h_k(X) = j)$$

The formula for calculating strength of Random Forest is as follows:

$$S = E_{X,Y}(mg(X,Y))$$

Some of the advantages of this algorithm is that it can handle large input of data, generates unbiased generalization error estimate, estimate missing data effectively and maintains accuracy efficiently [7] [27].

3.1.4 Stochastic Gradient Descent Classifier

Stochastic Gradient Descent (SGD) Classifier is one of the learning algorithms in machine learning which helps in mapping the data point to one of the classes. As mentioned earlier, the aim of the machine learning algorithms is to minimize the loss function which gives the inaccuracy of the predictions. Gradient Descent is a straight forward algorithm which performs different coefficient values and the cost function estimates their cost by comparing the predicted results with the actual results [8]. The algorithm tries different coefficient values to find the lower value which is updated using learning rate. However, GD is highly expensive as the cost is computed for entire training dataset for each iteration [8]. Therefore, SGD was introduced which updates the coefficient in several mini-batches of training set rather than taking the whole dataset at once, Figure 3.4. Hence, this algorithm helps in regularizing various machine learning models such as Support Vector Machine, Logistic Regression,

etc., that is, a penalty is added to the loss function using Euclidean norm (L2) or absolute norm (L1) to shrink the parameters of the model towards zero vector. The SGD algorithm is as follows:

$$\theta_j = \theta_j - \alpha(\hat{y}^i - y^i)x_j^i$$

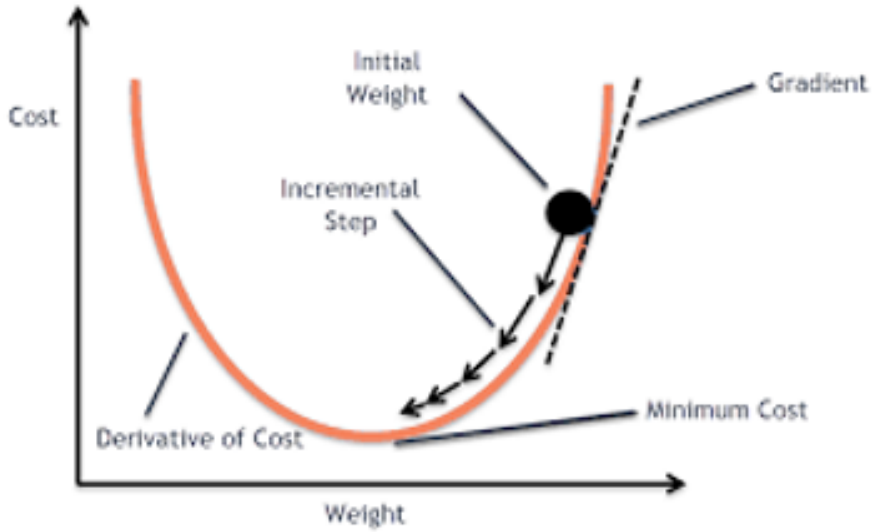

Figure 3.4: Stochastic Gradient Descent [23]

Linear Classifiers like Support Vector Machine and Logistic Regression with SGD training are SGD Classifier. The parameter loss changes its value according to these linear classifiers i.e. for Linear SVM, loss is set to hinge whereas for Logistic Regression, loss is set to log. The hyper-parameters used in this classifier for this research are max iter, that is, maximum number of epochs over the training data is taken as 1000 and tol, the stopping criterion of the training, is taken as $1e-3$.

3.1.5 Multi-Layer Perceptron Classifier

Multi-layer perceptron is the most used neural network in machine learning. It has two types of networks: feed forward and feed-back. A feed forward multi-layer network is the one in which the signals are transmitted in one direction within the network, that is, from input to output [44] [20] [39] Figure 3.5. This type of multi-layer architecture does not consist of any loops and the output of each neuron does not affect the neuron itself [44] [20] [39]. Whereas, feed-back network can transmit signals in both directions and are powerful as well as complicated. The complexity of decision region can be increased by introduction of several layers in the network. A one layer architecture generates decision

region in the form of semi planes, a two layer network generates convex regions and a three layer perceptron generates an arbitrary decision region [44] [20] [39]. A non linear activation function is used to power up a multi-layer perceptron. The most common activation functions are:

- **Logistic Sigmoid** - $f(s) = 1/(1+e^{-s})$
- **Bipolar Sigmoid** - $f(s) = (1-e^{-a*s})/(1+e^{-a*s})$

In multi-layer perceptron, the units are formed by adding the weighted sums of the inputs and then added to a constant. Then, the value is passed through the activation function.

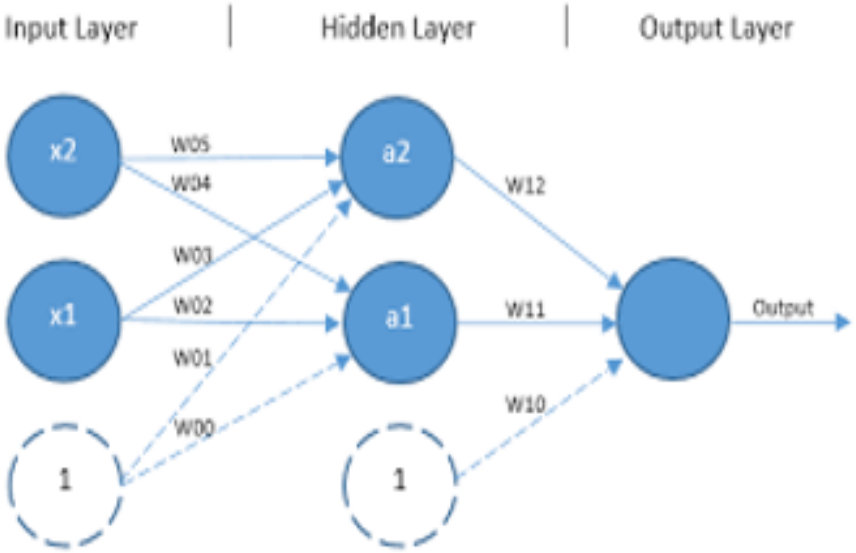

Figure 3.5: Structure of Neural Network [2]

Backpropogation Algorithm

Backpropogation is a learning algorithm in machine learning which uses gradient descent. The main idea is to minimize the error between network outputs and desired output [44] [20] [39]. The algorithm is as follows:

1. **Initialization** - This step includes the initialization of weights in the network with random values. The initial values should be small to avoid saturation which makes the learning slow.
2. **Training of the data** - In this, the weights are memorized and adjusted after each model in the training set and the end of the epoch of training, the weights will be changed only one time [44] [20] [39].

3. **Forward Propogation of the signals** - The formulas for outputs of neuron from the hidden layer, the output of the network and the error per epoch are as follows:

$$y_j(p) = f(\Sigma_{i=1}^{m} x_i(p).w_{ij} - \theta_j)$$

$$y_k(p) = f(\Sigma_{i=1}^{m} x_{jk}(p).w_{jk}(p) - \theta_k)$$

$$E = E + (e_k(p))^2/2$$

Where n is the number of inputs for the neuron j in the hidden layer, f is the sigmoid function and m is the number of inputs of the neuron in the outer layer.

4. **Backward Propogation of the errors** - The errors of the neuron in the outer layer, weights between hidden and outer layer, errors of the neuron in the hidden layer and weights between the input and the hidden layer can be calculated as follows:

$$\delta_k(p) = f'.e_k(p)$$

$$\Delta w_{jk}(p) = \Delta w_{jk}(p) + y_j(p).\delta_k(p)$$

$$\delta_j(p) = y_j(p).(1 - y_j(p)).\Sigma_{k=1}^{l} \delta_k(p).w_{jk}(p)$$

$$\Delta w_{ij}(p) = \Delta w_{ij}(p) + x_i(p).\delta_j(p)$$

where f' is the derived activation function and l is the number of outputs of the network.

5. **New Iteration** - When the epoch ends, if it satisfies the termination criterion, algorithm ends otherwise it goes back to step 2. The formula for updating the weights is:

$$w_{ij} = w_{ij} + \eta.\Delta w_{ij}$$

where η is learning rate.

3.1.6 Linear Discriminant Analysis Classifier

Linear Discriminant Analysis works best on the data in which within-class frequencies are unequal. This method guarantees maximum separability by maximizing the between class variance to within class variance in any data set as shown in Figure 3.6. There are two ways in which dataset can be transformed and classified using LDA [24] [4]:

- **Class-dependent transformation** - In this approach, main aim is to maximize the ratio of between class variance to within class variance to get the adequate class separability.
- **Class-independent transformation** - In this, each class is considered to be a separate class and maximizes the ratio of overall variance to within class variance.

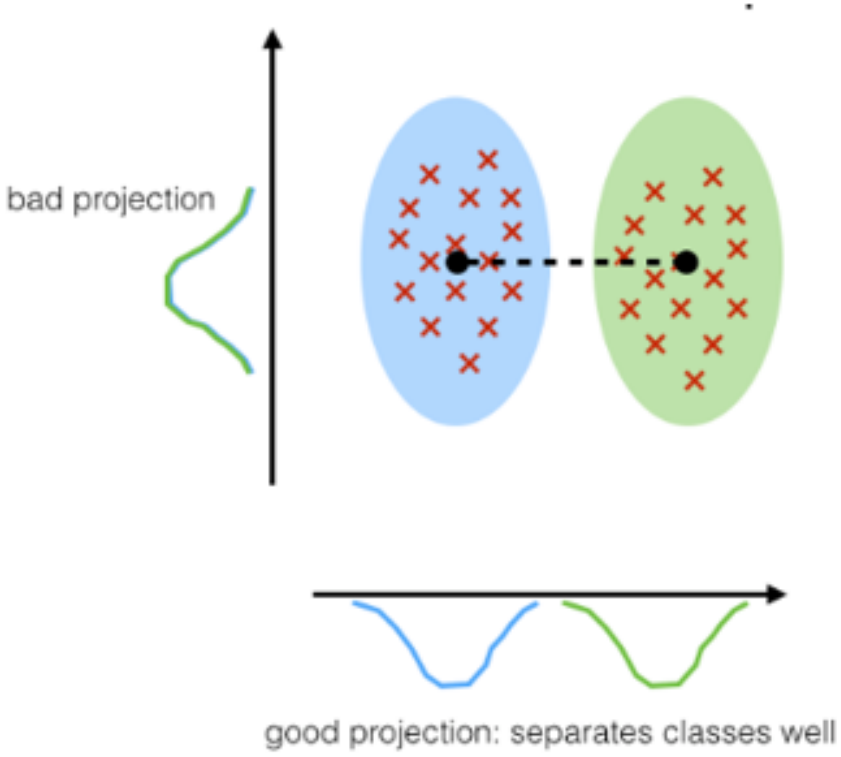

Figure 3.6: Graphical Representation of LDA [32]

The LDA classification algorithm is as follows [24] [4]:

1. The training data (set1) and the test data (set2) are generated.
2. The mean is computed for each of the datasets individually as well for the entire dataset.

$$\mu_3 = p_1 * \mu_1 + p_2 * \mu_2$$

where p1 and p2 are the probabilities of the classes and μ_1, μ_2 and μ_3 are the mean of classes.

3. The class separability is done using within class and between class scatter which can be calculated using following formulas:

$$S_w = \Sigma_j p_j * (cov)$$

where cov is covariance matrix with formula, $cov_j = (x_j - \mu_j)(x_j - \mu_j)^T$

$$S_b = \Sigma_j (\mu_j - \mu_3)(\mu_j - \mu_3)^T$$

For class dependent transformation, optimizing factor is calculated using:

$$criterion_j = inv(cov_j) * S_b$$

And for class independent is:

$$criterion = inv(S_w) * S_b$$

4. After all the transformations are completed, Euclidean distance is calculated to classify data points.

$$distn = (transformnspec)^T * x - \mu_{ntrans}$$

where μ_{ntrans} is the mean of the transformed data, n is the index of the class and x is the test set.

5. The smallest euclidean distance is considered to be the classification of test data.

3.2 Code Metrics Dataset

In machine learning, datasets are a collection of data instances, represented by rows that are sharing a common attribute, represented by columns. For learning of data by machine learning models, training data is fed into algorithms and is validated using the testing data which ensures the accuracy of the predictions. The larger the training data is, the faster the model learns and improves. The four types of data primarily used in machine learning are:

- **Numerical Data** - This is a type of measurable and quantitative data like length, height, cost, etc. It can be used for calculations by mathematical operations such as average, addition, sorting, etc. Discrete data is the one which consists of whole numbers and the numbers in a given range are continuous data.
- **Categorical Data** - This type of data defines the characteristics such as address, workplace, gender, name, nationality, etc. It helps in data analysis of large groups which share similar attributes.
- **Time Series Data** - Data that are indexed at a point in time are called time series data. It includes data of weeks, months and years. It can be distinguished from numerical data in a way that it has starting and ending points whereas numerical data does not.
- **Text Data** - It consists of different words, sentences, etc which can be analysed by machine learning models using various techniques such as text classification, word frequency, etc.

There are various platforms from where we can get machine learning data such as Google, Microsoft, Amazon Web Services, etc. However, for this dissertation, I collected the data myself and manually annotated it into a comma separated file that could be read by programming languages such as Python, R, etc. This dataset consists of code metrics generated from Verilog assignments of 18 students with 15 files per student. These assignments are collected from students of Trinity College Dublin who were in their second semester studying Verilog Hardware Description Language (VHDL).

3.2.1 Generating Code Metrics

Code metrics are nothing but a set of tokens extracted from the source code files, in this case, the verilog assignments submitted by the students. These tokens represent the characteristics of the programs written by a programmer and measures its effectiveness, style, system costs, reliability , flexibility and structure [21] [11]. There are various tools and techniques introduced to extract these code metrics including:

- **N-Grams** - These are the sequence of n items extracted from a text corpus such as words, letters, etc. In this method, documents are divided into list of substrings of length n with their frequencies [10]. It is a concept of Natural Processing Language. For example, United Kingdoms is a 2-gram.
- **TF-IDF** - It stands for Term Frequency-Inverse Document Frequency. It assigns weights to the words in a document and measures its importance in the corpus. It is mostly used for text mining and information retrieval.
- **ANTLR** - It is a parser generator which helps in reading and processing of programming languages. It also extracts words and syntax which is helpful in understanding the structure of the source codes.

However, the above mentioned tools and techniques either extract only words or do not work on Verilog programs but Java, Fortran, C, C++, etc. Moreover, in this research, besides identifiers and keywords, several other code metrics are needed to know the characteristics of the source codes such as spaces, tabs, number of words in a line, length of line, etc. Therefore, I used regex and many more libraries in Python programming language to write programs which extract these code metrics. Figure 3.7 shows the list of metrics that are used in this research with their description and tokens. Tokens are used to represent selected metrics which calculates the counts in one source code line such as Line Length, Line Words, Indent Space and Inline Space.

3.2.2 Training Dataset

Training dataset is the one which is used to build up machine learning models and run calculations to determine various coefficients. After fitting the training data into the models, they learn the pattern of the data which further helps in predicting or classifying the validation data (hidden data). In this dissertation, the dataset consists of a combination of code metrics and its frequencies, that is, the number of occurrences of each metric in every source code. The dataset has been annotated manually and contains source codes written by 18 students with 15 assignments per student. To build

Metrics Name	Coding System	Description
Line Length	LL	This metric measures the number of characters in one source code line.
Line Words	LW	This metric measures the number of words in one source code line.
Number of Line	Numeric	This metric measures the total number of lines a source code.
Inline Space Tab	IS	This metric calculates the whitespaces that occurs on the interior areas of non-whitespace lines.
Indent Space Tab	IN	This metric calculates the indentation of whitespaces used at the beginning of each non-whitespace lines.
Number of Spaces	Numeric	This measures the total number of spaces in a source code.
Number of Underscores	Numeric	This measures the total number of underscores present in a source code.
Identifiers with Underscores	Identifier_Name	This measures the number of identifiers which consists of underscores.
Number of Identifiers	Identifier_Name	Total number of identifiers present in the source code.
Number of Keywords	Keyword_Name	Total number of keywords present in the source code.

Figure 3.7: List of Source Code Metrics used for Plagiarism Detection in VHDL

the dataset, firstly, I counted the frequency of each code metric in an assignment and then, combined these tokens and frequencies that were labeled in a spreadsheet according with the metrics name. Table 3.1 shows an example of Line Length metrics with its frequencies in a program.

3.2.3 Validation Dataset

Validation data is a hidden dataset, excluded from the training set, used to check how well the predictions and classifications perform on wider set of data and gives the sense of false positives and false negatives which further helps in comparison of models. Although the validation set is similar to the training set, it does not contain the target values which has to be predicted or classified. In this case, validation set consists of 63 anonymous verilog assignments of the students. The overall sample structure of both datasets combined is shown in Figure 3.8.

Tokens	Token Frequencies
LL_5	10
LL_8	7
LL_15	6
LL_20	12
LL_28	1
LL_37	9
LL_46	11
LL_48	3

Table 3.1: An Example of Line Length Tokens and Frequencies

	Doc ID	Token and its Frequencies	Programmer
Training Dataset	Doc_1	(LL_5, 10), (LL_8, 7), (LL_28, 1)	Prog_1
	Doc_2	(LL_15, 6), (LL_8, 7), (LL_37, 9)	Prog_2
	Doc_3	(LL_15, 6), (LL_20, 12), (LL_48, 3)	Prog_3
Testing Dataset	Doc_4	(LL_5, 10), (LL_8, 7), (LL_15, 6)	?

Figure 3.8: Sample Structure of Dataset

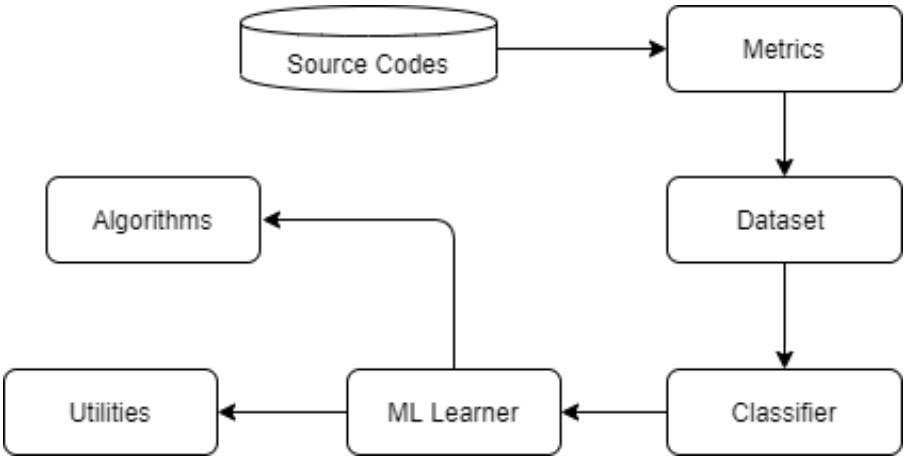

Figure 3.9: Architecture Diagram of the System

3.3 Implementation

Figure 3.9 shows the high level architecture of the plagiarism detection system being discussed in this dissertation. Each of the components mentioned in the figure is written using python programming language except for the source codes which are VHDL programs. Source codes are the raw data, that is, VHDL assignments collected from students. Metrics represent the tokens extracted out of the VHDL source codes such as line length, line words, etc. The code metrics generated are annotated into a spreadsheet which is then divided into two subsets, training and testing. Dataset represents the combination of all the code metrics with a target variable as programmer that has to be fed in into classifier. The classifier learns the pattern of the training data and build it into a machine learning model which is shown as algorithm. Utilities show the evaluation and testing of model using the testing data. The algorithms are evaluated on the basis of the classifications and where ever the classification is incorrect, it is considered to be a plagiarism between two files.

3.4 Evaluation

Machine learning models are able to provide prediction with high accuracy to be used as a real value by organizations. Although the main step is to train the model properly, generalizing on hidden data is another important aspect for models in machine learning pipeline. Various model performance evaluation techniques such as hold-out, cross validation, etc. as well as metrics to quantify their performances such as accuracy, confusion matrix, etc. have been introduced to handle this issue.The evaluation techniques and metrics which I have used to evaluate and compare the machine learning model in this dissertation are:

3.4.1 Model Evaluation Technique

Cross Validation is a method which evaluates the generalization ability of predictive models and prevents overfitting. It works on Monte Carlo Methods. There are two major challenges that have to be handled while training the data using machine learning, overfitting and underfitting. Overfitting happens when model learns the details and noise in the data which negatively impacts the performance of the model and underfitting occurs when the model does not capture well the relation between the variables [6] [26]. There are various techniques to handle these issues. However, the one which I have used while training my model is:

K-Fold Cross Validation

Cross validation is basically re-sampling of data in which no two test sets overlap [6] [26]. In k-fold cross-validation, the training data is divided into k disjoint subsets of equal sizes. This partitioning is done by sampling the data randomly without replacement. From these subsets, $k-1$ subsets are used as a training data and the k^{th} set is used as a validation set to measure the performance of the model as shown in Figure 3.10. This method repeats until each of the k subsets serve as a validation set. The cross validated performance can be measured by taking the average of all the k performance measurements.

$$\hat{e}_{cv} = \Sigma_{i=1}^{n}(y_i, \hat{f} - k(x_i))/n$$

Moreover, in some cases, cross validation is repeated with different k-fold subsets to reduce the variance of the performance measure [6] [26].

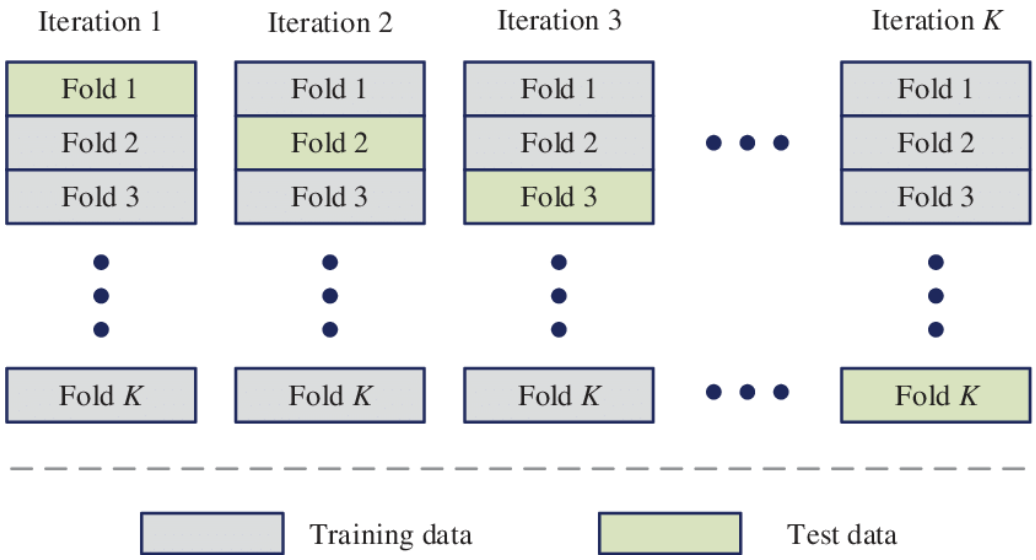

Figure 3.10: K-Fold Cross Validation [30]

3.4.2 Model Evaluation Metrics

These metrics are used to quantify the performance of the models and their choice depends on the machine learning tasks at hand. Some metrics like Precision and Recall can be used for multiple tasks. The metrics which I used to evaluate the models are:

Classification Accuracy

It is the most common evaluation metric for classification problems. Accuracy can be defined as the ratio of number of correct predictions and all the predictions made.

$$Accuracy = NumberofCorrectPredictions/TotalNumberofPredictions$$

For binary classification, the formula for accuracy is as follows:

$$Accuracy = TP + TN/TP + TN + FP + FN$$

where TP - True Positives, TN - True Negatives, FP - False Positives and FN - False Negatives. However, in case of imbalanced classes, it is not a fair measure to be optimized as it can be quite high, that is, it favours majority classes while ignores minority classes.

Confusion Matrix

Confusion matrix is one of the most informative performance measures a multi-class learning system can rely on [25] [14]. It provides information about the accuracy of classifier on one class and the confusion among classes, that is, the mistakes in classification of each class for other ones. They calculate the error measure according to class based costs of each misclassification. The interpretation of a confusion matrix involves diagonal elements representing the true predicted labels whereas all other values are the mislabeled classifications. The higher the diagonal values the better and more correct predictions. Moreover, confusion matrix are the most common tools for estimating goodness of a classifier in the imbalanced classes [25] [14]. However, in this research, confusion matrix is used as the criterion to determine the plagiarized files in the dataset.

F-Measure

F- Measure measures the accuracy of the test data and computes the score by considering the precision and the recall. It basically works according to Figure 3.11 where TP - True Positives, TN - True Negatives, FP - False Positives and FN - False Negatives.

Precision can be defined as the number of true positive results divided by total positive predictions.

$$p = TP/TP + FP$$

On the other hand, recall is the number of true predictions divided by total actual positives.

$$r = TP/TP + FN$$

Taking the weighted average of precision and recall leads to the F-Score [15].

$$F = 2 * ((precision * recall)/(precision + recall))$$

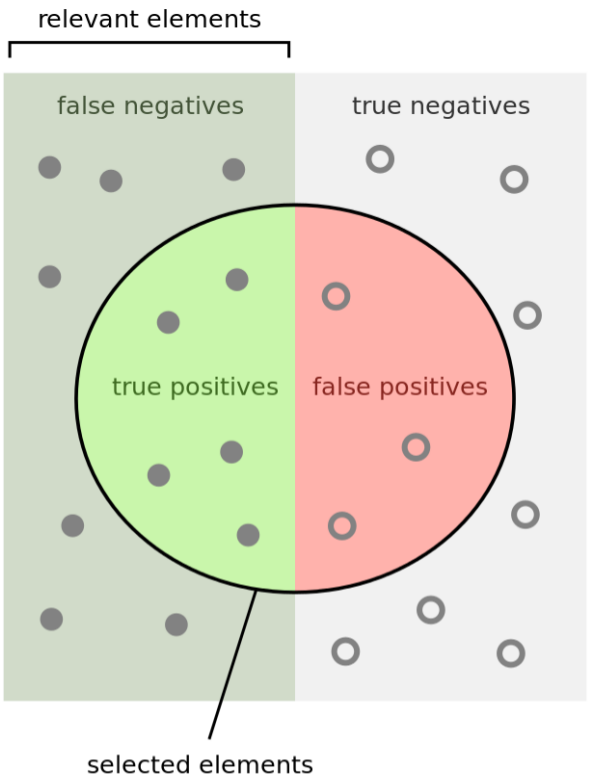

Figure 3.11: Representation of True Positive, False Positive, True Negative and False Negative [28]

Area Under Curve (AUC)

AUC, Area Under ROC Curve, is used to measure the classifier's ability for positive and negative class discrimination. AUC can be interpreted as a probability of a positive prediction ranked higher than the negative predictions. Its value ranges from 0 to 1 with 0 representing all wrong predictions and 1 represents all right predictions. AUC is one of the desirable performance metrics because [13]:

- Rather calculating the absolute values, it gives information about the ranking of the predictions.
- It does not varies with the classification threshold and calculates the quality of prediction of model.

Chapter 4

Results and Discussion

As this research focuses on two problem statements, this chapter includes, first, the results for best performing algorithm and then based on that algorithm, the results of detection of plagiarized files are shown.

4.1 Comparison of Trained Models

There are various techniques in machine learning which can be used to compare the performances of the trained models. However, the techniques which I am using in this dissertation are **Accuracy**, **Precision**, **Recall**, **F-Score** and **AUC** values. The test dataset consists of 63 anonymous verilog programs which are to be classified and their results were used to compute four major parameters which helped in finding the values of chosen evaluation metrics:

- **True Positives** - These are the positive files classified correctly by the model.
- **True Negatives** - Negative files which are classified correctly by the model.
- **False Positives** - Positive files incorrectly classified by the model.
- **False Negatives** - Negative files incorrectly classified by the model.

These outcomes help in evaluating various metrics like precision, recall, accuracy, etc. Accuracy is one of the most common evaluation metrics which gives us the percentage of correct classification made by a model. In this case, the accuracy of each algorithm came out to be as mentioned in Table 4.1.

According to Table 4.1, Stochastic Gradient Descent classifier is the best performing classifier among all the six models compared with 82.3% accuracy. However, for detection of plagiarized files, calculating accuracies is not enough to identify the best performing classifier so I further used AUC

Classifiers	Accuracies
SGDC	82.3%
MLPC	77.77%
Random Forest	74.6%
Decision Tress	57.14%
KNN	57.14%
LDA	49%

Table 4.1: Accuracy Table

metrics as the basis of comparison. This metric tells us the probability of positive classification ranked higher than the negative classification. The computed AUC values for all the classifiers are shown below:

Classifiers	AUC Values
SGDC	90%
MLPC	86.4%
Random Forest	86.46%
Decision Tree	77.8%
LDA	65.2%
KNN	53.24%

Table 4.2: AUC Table

Table 4.2 confirms that the Stochastic Gradient Descent Classifier has the best performance amongst all the classifiers used in this research. Moreover, precision, recall and f-score are the metrics which further helped in verifying outcome of the above metrics, that is, SGDC is the most accurate classifier. In the context of plagiarism detection, C. Arwin [3] mentioned in his research that precision represents the number of plagiarized files at some point in the classified list. As the precision increases, the detection becomes more accurate with fewer false positives. On the other hand, recall represents the number of plagiarized files out of all the plagiarized files in total. As the recall increases, there are less chances of plagiarized files escaping detection, that is, fewer false negatives. Furthermore, the classifiers with f-score closer to 1 is considered to have better accuracy than the classifiers with f-score closer to 0.

The Figures 4.1, 4.2 and 4.3 shows the average precision, recall and f-score for all the classifiers

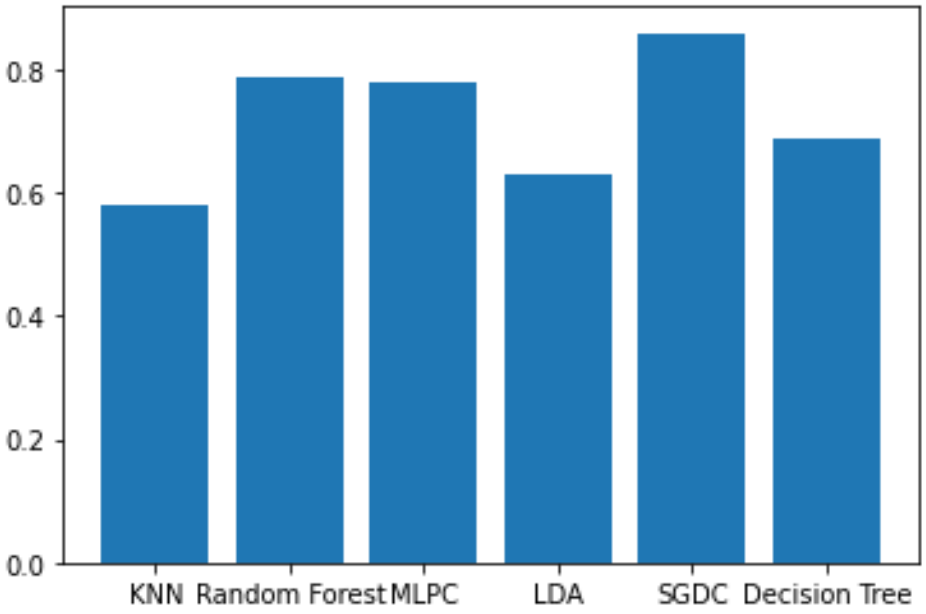

Figure 4.1: Precision Graph

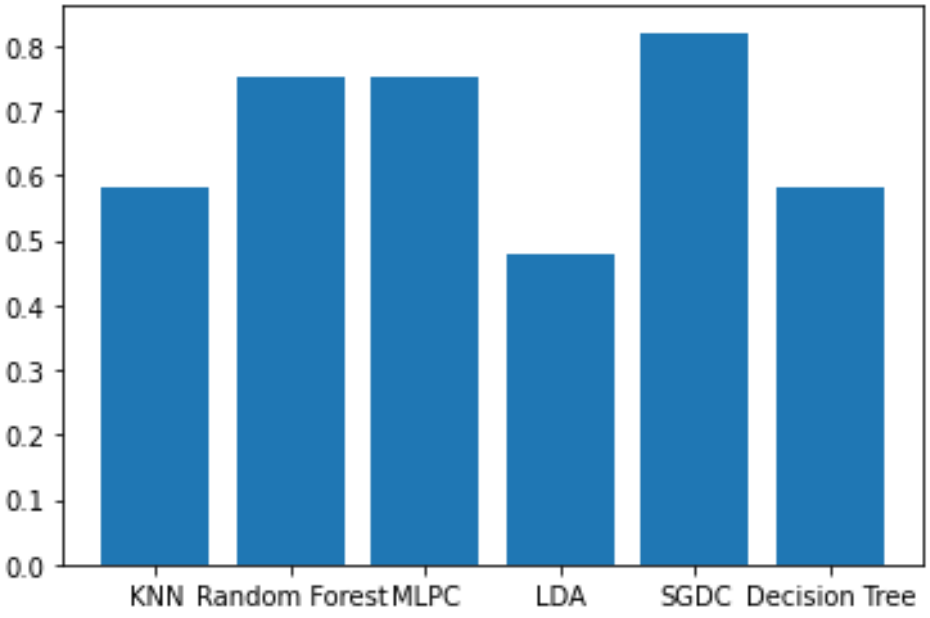

Figure 4.2: Recall Graph

and verifies that the Stochastic Gradient Descent Classifier has the best performance among all the six algorithms.

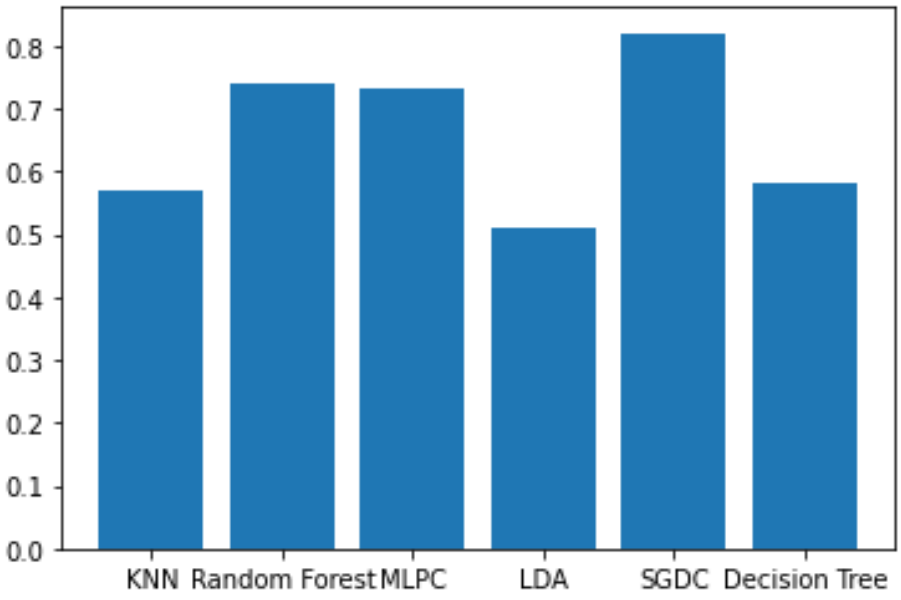

Figure 4.3: F-Score Graph

4.2 Detection of Plagiarized Programs

The main problem which has been focused in this dissertation is plagiarism detection in VHDL programs using machine learning. There are various ongoing as well as completed researches in this domain. However, most of them are either without using machine learning or are tested using other programming languages such as C, C++, Java, Fortran, etc. Therefore, I decided to come up with a machine learning technique which can be used to detect plagiarism in verilog hardware description language. I used confusion matrix as my evaluation metrics which helped me in knowing which files are similar to each other. For this problem, I have considered the results of only Stochastic Gradient Descent classifier as it performed the best i.e. the most accurate model among all the algorithms compared in this research. Figure 4.4 and 4.5 shows the confusion and the F-measure table, both generated using SGDC classifier.

The confusion matrix shown in Figure 4.4 is arranged in the same order as shown in the F-measure table. It shows the classification results of 63 anonymous assignments, four or three for each student. All the diagonal values shows the number of assignments correctly classified for each programmer whereas the column values show the predicted programmers and row values show the actual programmer. According to the Figure 4.4, first and second row show that all programs are classified correctly for Programmer 10 and 11, respectively. However, third row represents *Prog 12* which consists of four source codes to be tested. Among those four source codes, three were classified correctly

	#10	#11	#12	#13	#14	#15	#16	#17	#18	#2	#3	#4	#5	#6	#7	#8	#9
#10	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
#11	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
#12	0	0	3	0	0	0	0	0	0	0	1	0	0	0	0	0	0
#13	1	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0
#14	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	0
#15	0	0	0	0	1	2	0	0	0	0	0	0	0	0	0	0	0
#16	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	1	0
#17	0	0	0	0	0	0	1	2	0	0	0	1	0	0	0	0	0
#18	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0
#2	0	0	0	0	0	0	1	0	0	2	0	0	0	0	0	0	0
#3	0	0	0	0	0	0	1	0	0	0	2	0	0	0	0	0	0
#4	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0
#5	0	0	0	0	0	0	0	0	0	0	0	0	2	0	1	1	0
#6	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0
#7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0
#8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0
#9	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	3

Figure 4.4: Confusion Matrix

	precision	recall	f1-score	support
Prog_10	0.80	1.00	0.89	4
Prog_11	1.00	1.00	1.00	3
Prog_12	1.00	0.75	0.86	4
Prog_13	1.00	0.75	0.86	4
Prog_14	0.80	1.00	0.89	4
Prog_15	1.00	0.67	0.80	3
Prog_16	0.50	0.75	0.60	4
Prog_17	1.00	0.50	0.67	4
Prog_18	1.00	1.00	1.00	4
Prog_2	0.67	0.67	0.67	3
Prog_3	0.67	0.67	0.67	3
Prog_4	0.80	1.00	0.89	4
Prog_5	1.00	0.50	0.67	4
Prog_6	1.00	1.00	1.00	4
Prog_7	0.75	1.00	0.86	3
Prog_8	0.67	1.00	0.80	4
Prog_9	1.00	0.75	0.86	4
accuracy			0.83	63

Figure 4.5: F-Measure Table

whereas one seemed to be classified as *Prog 3*. This makes the code of programmer 12 suspicious to be similar to programmer 3. Once the similarity analysis is done and our algorithm gave the names of programmers who have some similarity in their programs, then I further inspected the suspicious files to determine the plagiarism in them.

```
submit Anon-012 Circuit_2.v
// Circuit_2

`timescale 1ns/100ps
`default_nettype none

module Circuit_2 (Out_1, Out_2, Out_3, A, B, C, D);
        output Out_1, Out_2, Out_3;
        input A, B, C, D;

        assign Out_1 = (A || !B) && !C && (C || D);
        assign Out_2 = ((!C && D) || (B && C && D) || (C && !D)) || (!A || B);
        assign Out_3 = ((A && B) || C) && D || (!B && C);
endmodule
```

Figure 4.6: Source Code - Programmer 12

```
submit Anon-003 Circuit_2.v
module Circuit_2 (Out_1,Out_2,Out_3,A,B,C,D);
output   Out_1,Out_2,Out_3;
input    A,B,C,D;

assign Out_1 = (A || (!B)) && (!C) && (C || D);
assign Out_2 = ((((!C) && D) || (B && C && D) || (C && (!D))) && ((!A) || B);
assign Out_3 = ((A && B) || C) && (D) || ((!B) && C);
endmodule
```

Figure 4.7: Source Code - Programmer 3

Figure 4.6 and 4.7 shows the verilog programs written by programmers 12 and 3, respectively. As both the programs came out to be similar except for the differences in spaces and brackets, it proves that the plagiarism occurred while writing this assignment. Likewise, there are many instances of plagiarism in the test data found by this technique introduced in this dissertation.

Chapter 5

Conclusion and Future work

5.1 Conclusion

In this dissertation, I discussed a machine learning technique using the best performing algorithm among six machine learning models that can be used for detecting plagiarism in verilog hardware description language programs. The two major problem statements focused in this research are how can we use machine learning for detecting plagiarism in verilog programs and which is the most accurate machine learning algorithm to detect plagiarism. The training and testing datasets generated for these problems consist of code metrics which were generated using python programming language. These code metrics include various features which helped in determining the structure of the verilog programs which, in turn, helped in finding programmers with similar files among all the programs.

The machine learning algorithms which were compared to find the most accurate model for plagiarism detection were K-Nearest Neighbour, Random Forest, Decision Tree, Stochastic Gradient Descent, Linear Discriminant Analysis and Multi-layer Perceptron. Among all these classifiers, **Stochastic Gradient Descent Classifier** came out to be the most accurate for plagiarism detection with **82.3%** accuracy and **90%** AUC value. For plagiarism detection, I used the results of Multi-layer Perceptron as it was the most accurate of all. Among 63 verilog assignments that were tested, 12 were confirmed as plagiarized and one of the instances is shown above in which programmer 12 and 3 have similar verilog codes with some differences in spaces and brackets.

5.2 Future Work

In this research, I have proposed a machine learning technique which helps in finding the similar verilog programs that have the possibility to be plagiarized as well as suggested the most accurate algorithm among six for detecting plagiarism. However, this research has a scope to be improved in

many aspects by including following things:

1. For now, this technique can be considered only as a tool for providing assistance to the instructors. It becomes difficult for machine learning algorithms to set absolute detection limits as there are many features in programs that influence plagiarism. Hence, it is important to have a manual evaluation of the suspicious programs by the instructors to sort their pairs. To make manual evaluation to minimal, more program features can be included in the training dataset.

2. In this research, the best accuracy is calculated for the multi-layer perceptron classifier, that is, 82.3%. However, this can be improved so that it gives us more accurate pairs of plagiarized files.

3. As this is still a prototype for the plagiarism detection technique, it can be developed for more real world use.

4. The dataset used in this research is manually annotated and due to time constraint, it consists of data of only 18 students. So, dataset can be increased both by adding the data of more students as well as increasing the code metrics such as text based and language dependent, which gives us the characteristics of programs more precisely. This can, in turn, help in increasing the accuracy of the algorithm.

5. This method can further be improved by using machine learning algorithms which provide us with the ranking of pairs of programs on the basis of similarity. For now, the algorithm is giving the result of comparison of two programs with maximum similarity. So, using alternate algorithms would help in the comparison of one file with all the other files.

6. This method does not capture the similarities of the programs if students have used any coding standards to write them. So, this issue can be considered and improved in the future.

Hence, these measures can be useful and help in improving this technique and use it for real world purposes.

Bibliography

[1] Giovanni Acampora and Georgina Cosma. “A Fuzzy-based Approach to Programming Language Independent Source-Code Plagiarism Detection”. In: *School of Science and Technology, Nottingham Trent University, Nottingham, U.K.* ().

[2] *Artificial Neural Networks – Part 2: MLP Implementation for XOr*. 2018. URL: https://www.mlopt.com/?tag=multilayer-perceptron.

[3] Christian Arwin and S.M.M. Tahaghoghi. “Plagiarism Detection across Programming Languages”. In: *Twenty-Ninth Australasian Computer Science Conference (ACSC2006)* 48 (2006).

[4] S. Balakrishnama and A. Ganapathiraju. “LINEAR DISCRIMINANT ANALYSIS - A BRIEF TUTORIAL”. In: *INSTITUTE FOR SIGNAL AND INFORMATION PROCESSING* ().

[5] Upul Bandara and Gamini Wijayarathna. “A Machine Learning Based Tool for Source Code Plagiarism Detection”. In: *International Journal of Machine Learning and Computing* 1.4 (Oct. 2011), pp. 337–343.

[6] Daniel Berrar. “Cross-validation”. In: *Encyclopedia of Bioinformatics and Computational Biology* 1 (2018), pp. 542–545.

[7] Gerard Biau. “Analysis of a Random Forests Model”. In: *Journal of Machine Learning Research* 13 (Apr. 2012), pp. 1063–1095.

[8] Shadi Diab. “Optimizing Stochastic Gradient Descent in Text Classification Based on Fine-Tuning Hyper-Parameters Approach. A Case Study on Automatic Classification of Global Terrorist Attacks.” In: *International Journal of Computer Science and Information Security (IJCSIS)* 16.12 (Dec. 2018).

[9] *Draw a program dependence graph with graphviz*. 2017. URL: https://stackoverflow.com/questions/46872521/draw-a-program-dependence-graph-with-graphviz.

[10] Michal Ďuračíka, Emil Kršáka*, and Patrik Hrkúta. “Current trends in source code analysis, plagiarism detection and issues of analysis big datasets”. In: *International scientific conference on sustainable, modern and safe transport* (2017), pp. 136–141.

[11] Matt G. Ellis and Claude W. Anderson. “Plagiarism Detection in Computer Code”. In: (Mar. 2005).

[12] J. A. W. FAIDHI and S. K. ROBIMOX. “AN EMPIRICAL APPROACH FOR DETECTING PROGRAM SIMILARITY AND PLAGIARISM WITHIN A UNIVERSITY PROGRAMMING ENVIRONMENT”. In: *Pergamon Journals Ltd* 11.1 (Mar. 1987), pp. 11–19.

[13] Peter Flach, Jose Hernandez-Orallo, and Cesar Ferri. “A Coherent Interpretation of AUC as a Measure of Aggregated Classification Performance”. In: *Proceedings of the 28th International Conference on Machine Learning* (2011).

[14] Garillos-Manliguez. “Generalized Confusion Matrix for Multiple Classes”. In: (Nov. 2016).

[15] Cyril Goutte and Eric Gaussier. “A Probabilistic Interpretation of Precision, Recall and F-score, with Implication for Evaluation”. In: *Proceedings of the European Colloquium on IR Resarch (ECIR’05)* (), pp. 345–359.

[16] Gongde Guo et al. “KNN Model-Based Approach in Classification”. In: *School of Computing and Mathematics, University of Ulster, Newtownabbey, BT37 0QB, Northern Ireland, UK* ().

[17] Daniël Heres. “Source Code Plagiarism Detection using Machine”. In: (Aug. 2017).

[18] *How can we use KNN, Machine Learning for classification of cars?* 2017. URL: https://www.quora.com/How-can-we-use-KNN-Machine-Learning-for-classification-of-cars.

[19] *Introduction to Programming Languages/Parsing*. 2020. URL: https://en.wikibooks.org/wiki/Introduction_to_Programming_Languages/Parsing.

[20] Alexander Iversen, Nicholas K. Taylor, and Keith E. Brown. “Classification and Verification through the Combination of the Multi-Layer Perceptron and Auto-Association Neural Networks”. In: *Proceedings of International Joint Conference on Neural Networks, Montreal, Canada,* (July 2005), pp. 1166–1171.

[21] Jeong-Hoon Ji, Gyun Woo, and Hwan-Gue Cho. “A Source Code Linearization Technique for Detecting Plagiarized Programs”. In: *ITiCSE’07, Dundee, Scotland, United Kingdom* (June 2007), pp. 73–77.

[22] Mark C. Johnson et al. “Gene Sequence Inspired Design Plagiarism Screening”. In: *American Society for Engineering Education Annual Conference Exposition* (2004).

[23] Divakar Kapil. *Stochastic vs Batch Gradient Descent*. 2019. URL: https://www.quora.com/How-can-we-use-KNN-Machine-Learning-for-classification-of-cars.

[24] Kang Soo Kim et al. “Comparison of k-nearest neighbor, quadratic discriminant and linear discriminant analysis in classification of electromyogram signals based on the wrist-motion directions”. In: *www.elsevier.com/locate/cap* (2011), pp. 740–745.

[25] Sokol Koco and Cecile Capponi. “On multi-class classication through the minimization of the confusion matrix norm”. In: *JMLR: Workshop and Conference Proceedings* (2013), pp. 277–292.

[26] Ron Kohavi. “A Study of Cross-alidation and Bootstrap for Accuracy Estimation and Model Selection”. In: *International Joint Conference on Artificial Intelligence* (1995).

[27] Vrushali Y Kulkarni and Pradeep K Sinha. “Effective Learning and Classification using Random Forest Algorithm”. In: *International Journal of Engineering and Innovative Technology (IJEIT)* 3.11 (May 2014), pp. 267–273.

[28] Chris L. *Evaluating ML Models: Precision, Recall, F1 and Accuracy*. 2019. URL: https://medium.com/analytics-vidhya/evaluating-ml-models-precision-recall-f1-and-accuracy-f734e9fcc0d3.

[29] Robert Lange and Spiros Mancoridis. “Using Code Metric Histograms and Genetic Algorithms to Perform Author Identification for Software Forensics”. In: *GECCO’07* (July 2007).

[30] Mingchao Li. *Tectonic discrimination of olivine in basalt using data mining techniques based on major elements: a comparative study from multiple perspectives*. 2019. URL: https://www.researchgate.net/figure/K-fold-cross-validation-method_fig2_331209203.

[31] Chao Liu et al. “GPLAG: Detection of Software Plagiarism by Program Dependence Graph Analysis”. In: *KDD, Philadelphia, Pennsylvania, USA* (Aug. 2006).

[32] Sambit Mahapatra. *(Linear Discriminant Analysis) using Python*. 2018. URL: https://medium.com/journey-2-artificial-intelligence/lda-linear-discriminant-analysis-using-python-2155cf5b6398.

[33] Raphael Njuguna. “A Survey of FPGA Benchmarks”. In: ().

[34] MATIJA NOVAK, MIKE JOY, and DRAGUTIN KERMEK. “Source-code Similarity Detection and Detection Tools Used in Academia: A Systematic Review”. In: *ACM Trans. Comput. Educ* 19.3 (May 2019).

[35] Karl J . Ottenstein. “AN ALGORITHMIC APPROACH TO THE DETECTION AND PREVENTION OF PLAGIARISM”. In: *CSD-TR 200* (Aug. 1976), pp. 30–41.

[36] Alan Parker and James O. Hamblen. “Computer Algorithms for Plagiarism Detection”. In: *IEEE TRANSACTIONS ON EDUCATION* 32.2 (May 1989), pp. 337–343.

[37] Bhaskar N. Patel, Satish G. Prajapati, and Dr. Kamaljit I. Lakhtaria. “Efficient Classification of Data Using Decision Tree”. In: *Bonfring International Journal of Data Mining* 2.1 (Mar. 2012).

[38] Fabian Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (Oct. 2011), pp. 2825–2830.

[39] MARIUS-CONSTANTIN POPESCU et al. “Multilayer Perceptron and Neural Networks”. In: *ISSN: 1109-2734* 8.7 (July 2009), pp. 579–589.

[40] James F. Power and John Waldron. “Calibration and Analysis of Source Code Similarity Measures for Verilog Hardware Description Language Projects”. In: *SIGCSE, Association for Computing Machinery, Portland, OR, USA* (Mar. 2020).

[41] Lutz Prechelt, Guido Malpohl, and Michael Philippsen. “Finding Plagiarisms among a Set of Programs with JPlag”. In: *Journal of Universal Computer Science* 8.11 (Nov. 2002), pp. 1016–1038.

[42] Thang Huynh Quyet. *Rule-Based Techniques Using Abstract Syntax Tree for Code Optimization and Secure Programming in Java*. 2014. URL: https://www.researchgate.net/figure/Abstract-syntax-tree-of-Euclide-function_fig1_300802898.

[43] Abilash R. *APPLYING RANDOM FOREST (CLASSIFICATION) — MACHINE LEARNING ALGORITHM FROM SCRATCH WITH REAL DATASETS*. 2018. URL: https://medium.com/@ar.ingenious/applying-random-forest-classification-machine-learning-algorithm-from-scratch-with-real-24ff198a1c57.

[44] Philippe Thomas. “Perceptron learning for classification problems”. In: *IJCCI’15 7th International Conference on Neural Computation Theory and Applications, NCTA* (Nov. 2015).

[45] Jagrati Valecha. *Building Blocks of Decision Tree*. 2018. URL: https://dimensionless.in/building-blocks-of-decision-tree/.

[46] Min-Ling Zhang and Zhi-Hua Zhou. “A Review on Multi-Label Learning Algorithms”. In: ().

[47] Justin Zobel. ““Uni Cheats Racket”: A Case Study in Plagiarism Investigation”. In: *Sixth Australasian Computing Education Conference (ACE2004)* 30 (2004), pp. 357–365.

Appendix A

Appendix

The code of this research has been uploaded on GITHUB and the link to it is: https://github.com/agupta1994/Dissertation_Plagiarism-Detection-in-VHDL-Programs-using-Machine-Learning.