Developing Machine Learning Coding Similarity Indicators for C & C++ Corpora

By

Ajinkya Kunjir

Master of Computer Science, Lakehead University, Thunder Bay, Canada

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Computer Science

In the
Department of Computer Science
Faculty of Science and Environmental Studies

LAKEHEAD UNIVERSITY
5th Term, 2020

Declaration of Committee

Name:	**Ajinkya Rajendrakumar Kunjir**
Degree:	**Master of Computer Science**
Thesis title:	**Developing Machine Learning Coding Indicators for C & C++ corpora**
Committee:	**Chair:** Dr. Jinan Fiaidhi [Professor, Computer Science Department]
	Dr. Jinan Fiaidhi Supervisor [Professor, Computer Science Department]
	Dr. Sabah Mohammed Committee Member [Professor, Computer Science Department]
	Dr. Zuhoor Al-Khanjari External Examiner [Professor, Department of Computer Science, College of Science, Sultan Qaboos University]

Abstract

The digital data in this modern world is vulnerable to copying, altering and claiming someone else’s work as their own. Performing the same activity in programming assignments can be referred to as source-code theft or e-plagiarism. Despite years of efforts, the already existing similarity detection engines perform pretty well in detecting plagiarism for novice programmers, but provides insufficient results when a student uses complex and smart plagiarism hacks such as word substitution, structure change, line spacing placeholder comments. This thesis research aims to deliver an assistive forensic engine named ‘SimDec’, for the evaluators to help detect similar assignments to address the aforementioned issues. The system's primary objective is to aid the assignment evaluators to get closer to the code thieves and abide by the university's dishonesty regulations. The forensic engine has been developed in Java programming language to detect C and C++ source code's similarities. The research has been split into two modules labelled as ‘software forensic engine development’ and ‘Similarity level classification with machine learning’. The proposed system has a workflow of three stages starting with lexical analysis, tokenizer customization and the final stage displaying similarity percentage and the corresponding level of ‘Low’, ‘Average’ and ‘High’. The combination of similarity algorithms integrated in the engine are Levenshtein distance, Jaro & Jaro-Winkler measure, Dice coefficient and Cosine similarity. The workflow of lexical analysis and implementing the set of similarity measures on token categories is defined as the first module. The machine learning algorithms selected for performing the classification task are multi-class SVM, logistic regression and a simple neural network. In this second module, the data gathered and generated by the similarity detection engine is fed to the ML algorithms to train the models and make them efficient for predicting the plagiarism or similarity level of newly entered data. This hybrid approach would be impactful in reducing the time complexity and processing speed for the software engine.

Keywords: Plagiarism, lexer, ANTLR, distance algorithm, similarity, forensic engine

Dedication

I hereby declare that this dissertation/thesis entitled “MACHINE LEARNING CODING SIMILARITY INDICATORS FOR C & C++ CORPORA” is a bonafide and genuine research work carried out by me under the guidance of Dr. Jinan Fiaidhi, Ph.D. (in Computer Science (Brunel, UK, 1986).

Date:

Place: Lakehead University

Ajinkya Kunjir

Acknowledgements

It is a genuine pleasure to express my deep sense of thanks and gratitude to my mentor, a great teacher and my supervisor Dr. Jinan Fiaidhi, a full-time professor and the graduate coordinator of the Ph.D. program in Biotechnology at Lakehead University, Thunder Bay. Her dedication and interest, above all her overwhelming attitude to help her students, have been solely and mainly responsible for completing my work. Her timely advice, meticulous scrutiny, scholarly advice and scientific approach have helped me greatly to accomplish my task.

I owe a big thank you to Dr. Fiaidhi for recommending me to research this domain of computer science and keep keen interest with me on the same at every stage of my research. Her prompt inspirations, timely suggestions with kindness, enthusiasm and dynamism have enabled me to complete my thesis.

I want to thank all the faculty members of computer science, Lakehead University, for their constant guidance in the subjects and areas outside academics throughout my program's tenure. I am incredibly thankful to my parents for having faith in me and providing the necessary strength and suggestions during my stay in Canada.

Table of Contents

List of Tables

List of Figures

List of Acronyms

LU	Lakehead University
LAC	Library and Archives Canada
ANTLR	Another tool for Language Recognition
SIMDEC	Similarity Detection Forensic Engine
ML	Machine Learning
SVM	Support Vector Machine
NN	Neural Networks
CNN	Convolutional Neural Networks
ReLU	Rectified Linear Unit
SGD	Stochastic Gradient
IEEE	Institute of Electrical and Electronics Engineer
LL1	Left-to-right, leftmost derivation

Preface

This research basis initially stemmed from my passion for developing better program analysis and knowledge representation methods. As the world moves further into the coding age, generating vast amounts of source code information, there will be a greater need to access original content for innovation and not reuse the labelled content. How will we use our logic to create our code? It is my passion to investigate this issue and develop tools to break down barriers of accessibility for future generations. In truth, I could not have achieved my current level of success without a strong support group. First of all, my parents, who supported me with love and understanding. Secondly, my committee members and colleagues, each of whom has provided patient advice and guidance throughout the research process. Thank you all for your unwavering support.

Chapter 1.

Introduction

1.1 Problem Context

According to the University Academic Dishonesty Regulations, stealing others' ideas or not giving credits to the original author is considered misconduct and an act of plagiarism. Source-code theft often happens in the assignments given to the students in their academia. Since it is difficult to track down thousands of assignment documents while reviewing one, there arises a need for an assistive tool or web application, which aids the teaching assistants, or the professors get closer to the sophisticated code thieves.This research study aims to explore several options for tracking similarities betw een 'N' number of targets, also known as 'documents,' in response to this problem.
Several algorithms in the basket from which a few or a combination of string similarity algorithms could be constructed according to the need. The algorithms are chosen from a wide range of distance similarity algorithms as well. The model can be trained and tested on a dataset of choice. The research will consider vital resources, less data consumption algorithms to mitigate some or all of the problems noted above or in the later sections of the thesis.

1.2 Thesis Statement

The available source code similarity detectors provide unsatisfactory results when students use complex strategies such as word substitution or reordering programming constructs. This thesis research proposes an assistive forensic engine for the professors and teaching assistants to evaluate the similarities in the student's assignments to overcome the above-mentioned challenges. This research's primary objective is to help the evaluators get closer to the sophisticated code thieves and abide by the university's academic dishonesty regulations. The proposed forensic similarity detection engine named 'SimDec' has a constructive methodology that is specially designed for studies where C and C++ programming languages are majorly used in academic assignments.

After selecting the ATM (Attribute counting metrics), the system implementation is divided into two phases, where phase one consists of lexical analysis, tokenizer customization and visual representation on web GUI. The similarity elements and observations recorded could be represented to the evaluators in the form of visualizations for ease of understanding and efficient decision-making. The second phase involves of performing supervised machine learning algorithms such as multi-class SVM, logistic regression and neural networks on the systems data for executing classification / prediction task for newly entered data. The algorithms after successful training could be capable of classifying the data records according to plagiarism level such as 'low, 'medium' and 'high' and result in reduced computation time.

1.2.1 Methodology

To develop a similarity detection forensic engine based on lexical analysis, a systematic approach was followed. This allowed rigorous testing and implementation of various mathematical equations and computations. The steps undertaken in developing the system in the methodological approach are given below as follows:

- Setting up the ANTLR tokenizer with the C and C++ grammar for performing lexical analysis. Implementation of ANTLR tokenizer and integrating it to function with the platform of choice. Initially, eclipse IDE was chosen as the development environment but because of incompatibility issues, there was a switch made to Intellij IDEA later.
- Extract and encode the downloaded IEEE Homework Programming dataset (IEEE Dataset weblink) into the java source code snippet.
- Secure the full-stack communication connectivity for java code with MySQL database and front end.
- Ensure the comparison of one source code assignment with others in the folder and breakdown formation for lexemes.
- Configuring smart decision making in the code to refer to the extension of the file (whether C or C++)

We are implementing the system's code starting from lexical analysis to token grouping followed by applying distance similarity algorithms on the token groups for computing score. Implementation of distance-based, token-based and sequence-based algorithms on the lexemes of the source codes for recording observations. The scores are generated as results for all algorithms, and a conclusion is made by observation of all scores to justify if the two source code assignment files are similar / plagiarized or not.

- We are storing the experimentation results in the database for analytics and knowledge representation in the form of visualizations. Implementation of a web graphical user interface to display all the scores in form of visual representations like charts and graphs for ease of understanding.

- Second module of the thesis research – Implementation of supervised and unsupervised algorithms on the generated dataset from the system for advanced machine learning indicators. This module has been experimented on a different platform for ease of coding and low latency.

In addition to this section, a serial java code was developed to validate the new features' logic at each step of development. The below-listed platforms and software plugins were used for the successful implementation of the forensic engine:

Intellij IDEA

Integrated development environment developed by jet Brains on the pillars of java programming language. This IDE provides coding assistance for other languages such as C++, C, C#, PHP, web with community edition, and various plugins support.

ANTLR

ANTLR is a lexer and a parser generator for reading, processing and executing text files. ANTLR has programming language support for all languages such as C, C++, Java, python, Ruby, and Grammar Construction.

JAVA

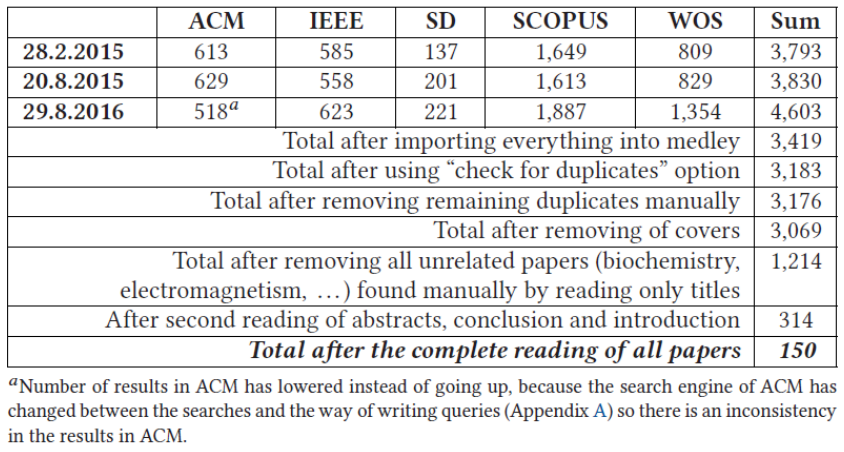
Java is a class-based object-oriented language used to design the similarity detection engine and encode all the other plugins. Being architecture-neutral and quite flexible, Java is the best-suited language for building software systems

Appendix A and B gives more details about the coding environment and integration systems mentioned in this research. The implementation of these different tasks and the analysis of the results using distance similarity algorithms are presented in this paper's following sections.

1.3 General Thought on Prior Work

Previous exciting research in the past 13 years about source code plagiarism and several software have been developed for the same. Matija Novak [2] in the latest 2019 research on source code plagiarism recited a systematic review on the plagiarism and source-code detection methods, obfuscation methods, definitions of plagiarism and algorithms used in the existing tools. The popular databases such as SCOPUS, ACM, IEEE, SD, and WOS were filtered with in-depth queries for searching plagiarism relevant papers. The results underwent several checks such as Medley tool passing, checking for duplicate options, removing duplicated manually, after removing covers and all others you can see in the below-mentioned figure:

Table 1: Latest 2019 research- Matija Novak [2] Database Querying



	ACM	IEEE	SD	SCOPUS	WOS	Sum
28.2.2015	613	585	137	1,649	809	3,793
20.8.2015	629	558	201	1,613	829	3,830
29.8.2016	518[a]	623	221	1,887	1,354	4,603
Total after importing everything into medley						3,419
Total after using "check for duplicates" option						3,183
Total after removing remaining duplicates manually						3,176
Total after removing of covers						3,069
Total after removing all unrelated papers (biochemistry, electromagnetism, …) found manually by reading only titles						1,214
After second reading of abstracts, conclusion and introduction						314
Total after the complete reading of all papers						***150***

[a]Number of results in ACM has lowered instead of going up, because the search engine of ACM has changed between the searches and the way of writing queries (Appendix A) so there is an inconsistency in the results in ACM.

The authors mentioned that 120 out of 150 finalized papers report some new tool or algorithm for similarity detection. After deep research, a 'table of tools' in the article was concluded, which specified the year of birth, last year of upgrade, number of times compared, number of times used and developed. The common point to note here is that not a single tool invented is available for public use (Not FOSS). There were only five tools, which were compared at least two times, out of which MOSS, and JPlag are the top-two tools, followed by the others shown in the table embedded in the figure given below:

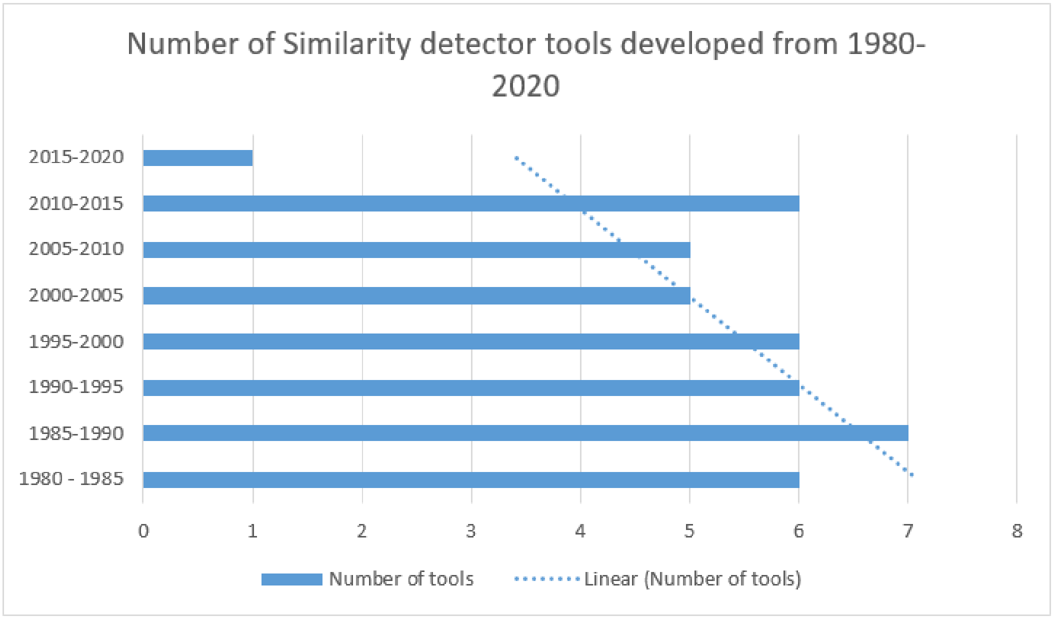


Figure 1: Number of Similarity detectors developed from 1980-2015

Table 2: Overview of top plagiarism detection tools according to the SEO of internet

Tools	Last Year	First Year	Compared	Used	developed
SIM- Grune	2014	2010	4	2	NA
Plaggie	2016	2006	6	0	1
Sherlock-Warwick	2016	1999	4	4	1
MOSS	2016	1999	29	9	NA
JPlag	2016	2002	37	5	1

Referring to the table and figure above, the existing tools such as MOSS, JPlag and Plaggie have not been updated since 2016 and this acts as a primary motivation for potential innovation in the proposed forensic engine. Adding to the already existing tools' flaws, no graphical user interface was proposed or built for the previous devices. A graphical UI presented with our forensic engine eases the process of knowledge representation in the form of visualizations, displaying results, analytics and clusters of plagiarized and unique student source code assignments. A few things not present in the previous tools add a novelty feature to the proposed system. Not to mention the thesis of Daniel Heres in 2017 [3]. He built a system for source code similarity detection based on mathematical measures such as n-grams, tf-idf and cosine similarity. ‘InfiniteMonkey’, was the system built that could identify suspicious similarities between the source code documents using two methods. The second part mostly focuses on applying complex neural network models on the synthetic dataset created from the source code documents parameters. The methods were compared on several datasets and accuracy was measured. It was concluded that the deep neural network model does not generalize well to the evaluation tasks. A few visualization techniques are also displayed using the tool 'InfiniteMonkey'.

1.4 Need for the System

We often hear about the words 'Plagiarism' and 'Prevention' from the teachers or professors at high school as a pre-measure before submitting any assignment or homework whatsoever. 'Plagiarism' can simply be defined as an act of cloning someone's idea or concept and sticking it to our own with or without others' consent. When it comes to a large of the class, it is very complicated to traverse and keep checking all the assignments while checking any one. One can't just keep track of all the programming methods, concepts, and logic students use in their assignments. There is a need for an assistive tool that can help the teachers or their assistants check the assignments and visualize the contents with analytical results. Source code duplication has been increased over the years and is problematic for the future of innovations. The similarity detection system proposed in this thesis research is a mixture of multiple computer science streams such as software engineering, program analysis, programming language constructs, compiler design theory and web technology. The three-tier forensic engine makes use of a tokenization – lexical analysis approach in which a large dataset of C and C++ source codes will be under experimentation. The system will have a code for comparing two files under an interactive loop, which will keep comparing one source code assignment with others in the folder and then go on to the next one. The distance similarity algorithms such as Levenshtein distance, Jaro and Jaro-Winkler, cosine similarity and dice coefficient are applied to compare source code files. The results deduced from the system will get the evaluator closer to the plagiarized assignments to make an intelligent decision. Considering all the advantages of the forensic engine, there arises a need for such a system that could detect similarities in assignments consisting of object-oriented programming constructs. Succeeding the current chapter, chapter 2 delivers a critical survey on the researches that have been accomplished previously. Moreover, the next chapter describes the evolution of plagiarism detection software's over the years 2007 –

2019 and different categories of plagiarism tools. Chapter 3 illustrates the implementation of the forensic similarity detection engine labelled ‘SimDec’ along with the details of system working, three-layer architecture and software engineering process. Chapter 4 and 5 sheds light on mathematical similarity algorithms used for similarity detection and supervised ML algorithms implementation on the file-comparison data generated by the system. Chapter 5 will discuss the details of ML algorithms with their performance evaluation parameters such as accuracy, loss and a comparative study amongst them. The thesis is concluded with the conclusion and future works in chapter 6. Appendix A after the last chapter shows the necessary steps for installing ANTLR jars and plugins in the development environment with the help of screenshots. Appendix B shows the main ‘file comparison loop’ java programming language source code in a structured format.

Chapter 2. Scientific Background

2.1 Similarity Detection as a Solution

Source code plagiarism in the education sector is a grave concern for educators and students because of online assignment submissions and dividing the marking work between multiple teaching assistants to evaluate the students. The incapability of comparing one assignment with others has led to an increasing case of plagiarism in education. From the recent survey from teachers across schools in the world, it was mentioned that the novice programmers have evolved with time and due to shuffling code blocks, adding comments and tapping space frequently. Such obfuscation techniques make the evaluator impossible to detect similarities in multiple assignments, and therefore, there arises a need for plagiarism checker software for the evaluators. The similarity detectors could also serve to protect the 'Copyright Infringement' policy under copyright law. Student copying another student's work can face guilty charges of going against the anti-plagiarism rule of the university's dishonesty regulations and could face suspension/ temporary term extension. The guilty charges rained on the student for submitting plagiarized work can hamper the progression early and reflect negatively on the school transcript.

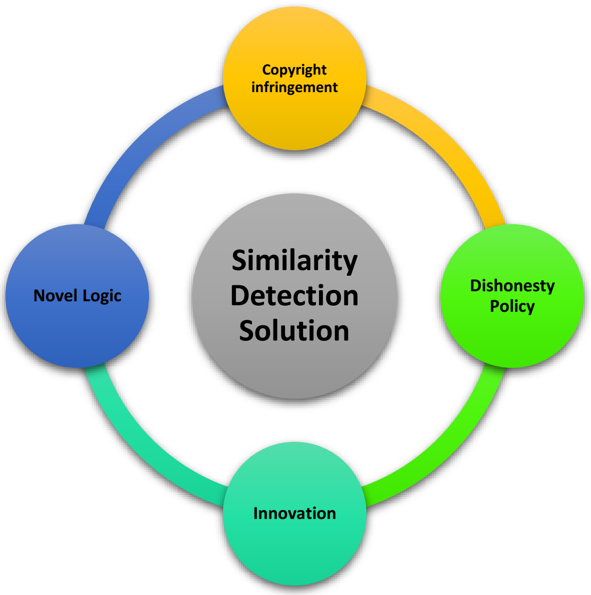


Figure 2: Solution elements of similarity detection

2.2 Evolution of Plagiarism Detector Software

Several researches have shown traces of similarities between the student assignments where digital submission platform is involved in the past few years. In the year 1986, J. A. W. Faidhi and S. K. Robinson [4] in their research provided an in-depth analysis of program similarity and reported plagiarism for 'Pascal' programming language. The authors mentioned two metrics (ATM: Attribute Counting Metrics), such as the first metric was intended towards a novice programmers approach, and the second one focused more on program structures and how programming blocks can be modified or altered by a sophisticated programmer. Al-Khanjari et al. [5] in their research development on a software entitled 'PlagDetect' referred to various ATM's before designing the final system for finding similarities between java source codes. As discussed in chapter 1, SM's (Structure methods) perform efficiently well than ATMs as they deal with spotting similarities by observing the program structures such as loops (for, if, while), class structure, functions identifier positioning. PlagDetect tool was based on a similarity coefficient and a combination of ATM's and equivalence ratio for investigating java assignments. The other invented similarity detectors such as YAP3, MOSS, Plaggie and Deckard make use of tokenization or winnowing with string similarity or distance similarity measures such as n-grams, cosine and Karp Robin GST (Greedy String Tilling). Other Software tools such as ccfx and iclones make use of suffix tree matching for finding similarities between parse trees. Given below, the table 1 provides the audience with a detailed comparison of all categories of similarity detectors such as plagiarism detection tools (PD), clone detector tools (CD) and others (O), which also includes compressors and mini-tools. In addition to comparing the tools with their similarity measurement calculation, we have added details, default parameters and year of the invention along with the research paper reference in the columns of the same table.

Table 3: Table displaying tool names with similarity calculation method, year of invention and reference to the publication

Tool category	Similarity Calculation	Details	Year and Reference
(PD) simtext	Tokens and string alignment	Min. run size	1999 Gitchell and Tran [6]
(PD) Simjava	Tokens and string alignment	Min. run size	1999 Gitchell and Tran [6]
(PD) Sherlock	Digital signatures	Chain length, zero bits	2002

			Pike R and Loki [7]
(PD) Jplag-text	Tokens, GST(Greedy String Tiling), Karp-Rabin	Min. no. of tokens	2002 Prechelt et al. [8]
(PD) Jplag-java	Tokens, GST(Greedy String Tiling), Karp-Rabin	Min. no. of tokens	2002 Prechelt et al. [8]
(CD) ccfx	Tokens and suffix tree matching	Min. no. of tokens	2002 Kamiya et al. [9]
(CD) YAP	Tokens, GST(Greedy String Tiling), Karp-Rabin	Tokenization and GST matching	1996 Michael J. Wise [10]
(PD) plaggie	N/A (Not disclosed)	Min. no. of tokens	2006 Ahtanein et al. [11]
(CD) deckard	Characteristic vectors of AST optimized by LSH	Min. no. of tokens Sliding window size Clone similarity	2007 Jiang et al. [12]
(CD) nicad	TXL and string matching (LCS)	Percentage of unique code Min. no. of lines Code abstraction Variable renaming	2008 Roy and Cardy [13]
(CD) iclones	Tokens and generalized suffix tree	Min. of tokens	2009 Gode and Koschke [14]
(O) cosine	Cosine similarity from machine learning library	N/A	2011 Pedregosa et al [15]
(O) fuzzywuzzy	Fuzzy string matching	Similarity calculation	2011 Cohen [16]
(O) ngram	Fuzzy search using n-gramme	N/A	2012 Poulter [17]
(CD) simian	Line based string comparsion	Min. no. of lines Ignoring variables, whitespaces, identifiers	2015 Harris [18]
(O) Difflib	Gestalt pattern matching	Ignoring whitespace, autojunk heuristic	2016 Python Software Foundation [19]
(O) diff	Equation	N/A	2016
(O) bsdiff	Equation	N/A	2017
(O) jellyfish	Approximate and Phonetic String matching	Edit distance algorithm	2016 Turk and Stephens [20]
(C) 7zncd	NCD with 7z	Compression level	N/A
(C) Bzip2ncd	NCD with bzip2	Compression level	N/A
(C) gzipncd	NCD with gzip	Compression level	N/A
(C) xz-ncd	NCD with xz	Compression level	N/A
(C) icd	Regular NCD (Normalized Compression Distance)	Compression level, block size	N/A
(C) ncd	Regular NCD	Compression level	2015,Cilibrasi et al. [21]

The evolution of similarity detection engines ranges from the one's using ATM's to one using SM methods. There are other systems mentioned in the table above, which uses the suffix tree-matching algorithm.

2.3 Types of Plagiarism Tools

2.3.1 Inter-Document Plagiarism with Internet Context

There are two different ways of detecting similar content in student assignments. The existing tools can be classified into two different categories: ones that compare the source codes with each other in one directory and the other category of tools compare the source code with all others available on the internet. The phenomenon occurring in the second category of tools is called as 'Inter-Document Plagiarism with internet context'. The software tools are web applications with register, login, and purchase feature where the user gets to upload the source codes to the application's cloud storage and then the same document is checked for similarities with other forms on the internet.

The plagiarism tools for this category, which are available for public use on the internet, are as follows:

- PlagScan[1] – PlagScan is an online plagiarism checker that uses an undisclosed algorithm to check the inserted or uploaded text by automatically scanning billions of texts online. The tool is not an open-source software under GPL and is for private or subscribed use only. The tool offers an option to detect differences in writing styles, checking the source code against billions of source codes available on the internet, ghostwriting detection, stylometry analysis. PlagScan is easy to integrate with a learning management system (LMS) and allows import from other sources such as Google, Dropbox and Microsoft OneDrive. PlagScan is currently working on text translation and comparing the text from the documents with other texts online.

- CopyLeaks[2] – Copyleaks is an inter-document plagiarism checker with AI capabilities for detecting similar contents in text documents and other media. Plagiarism checking facility is available in multiple languages and equipped with machine learning technology and extensive multi-layered search capabilities. Copyleaks has many advantages over PlagScan. A few to include would be aesthetic report making, multiple language detection, and comprehensive source code plagiarism and integration ability with other existing IDE’s.

- UniCheck[3] – Unicheck plagiarism reporting software prefers results over numbers by combining technological excellence and initiative design. Unicheck is exclusively made for to be used in the education sector. Unicheck is capable of handling extensive data at a time due to cloud storage and other features. Just like grammarly, Unicheck has the ‘recommendation’ or ‘advice’ giving feature where there are alternative word being suggested by the software to replace the current word if it’s repetitive in the text. With unicheck, the evaluator can get an accurate similarity score and see the amount of borrowed text. Unicheck has customizable search settings, real-time checking features, smooth integration and setup, and statistics for students.

- Codequiry[4] – Codequiry is the best source code detection platform available for public use in today’s world. There is no other competitive web application to stand against advantages of Codequiry. Codequiry looks beyond the cosmetics, finding similarities in logic patterns and unique style of code. Codequiry is possible of dodging minor obfuscation techniques such as whitespaces, comments, and function names. User has the facility of choosing the programming language for similarity detection such as java, c, c++ or python and then can upload a bunch of files to check the similarity. Codequiry is the only existing similarity detector that performs intra as well as inter-document plagiarism. The uploaded files can be compared with other billions of files on the internet and one another, depending on the option chosen by the user.

2.3.2 Intra-Document Plagiarism

- Codequiry[4] - As mentioned in the section above, Codequiry is exceptionally efficient for intra-document plagiarism. The software gives out detailed reports with analysis and visualization to inform the statistics (Results and numbers). Codequiry is well known for peer checks with the internet against 20 billion documents, a 2D graph of student similarity by distance and pie charts of source codes used in the code. The only disadvantage of Codequiry that excludes it from the outer world and innovation is proprietary usage. The individual pricing for educators or students is 29$ per month and the educational institutes' custom amount.

- MOSS – MOSS stands for ‘Measure of Software Similarity’ and is an automatic system for determining source codes' similarity [22]. From 1994 to today’s date, MOSS is being utilized to top-notch quality by some educational institutions to spot similarities between source code documents. The algorithm used by MOSS is ‘winnowing’ and involves of specific n-gram technique measures. MOSS is just an assistive tool that gets the evaluator closer to the cheaters, just like our proposed system. MOSS is available as a Windows GUI and lets users upload files by specifying the subfolders' directory path. MOSS simply computes and gives out a matching number of lines in front of the two filenames as output. A user could then click on the filenames and view the source code comparison highlighted in red or blue color. Relating it closely with beyond compare, MOSS is a child of the principles of beyond compare tool.

2.4 IEEE Homework Programming Dataset

Generating source code dataset using artificial techniques is a challenging task and indirectly reflects various realistic situations. Referring to figure 4, the new homework programming dataset is presented in this research to work with the proposed system, as there is a lack of description of standard datasets in existing researches. The ‘Programming Homework Dataset for Plagiarism Detection’ was uploaded on IEEE-Dataport by Vedran Ljubovic, University of Sarajevo [23]. The dataset is developed from the students' assignments for the subject – Introduction to C in one semester and assignments of C++ in other for the year 2016 and 2017. All the final source codes submitted by the students are available at the [5]website and on AWS for comparison by the already existing plagiarism detection tools like JPlag, YAP3, MOSS and PlagDetect. The homework assignment zip extract consists of full traces of student activity and keystrokes generated by setting the IDE to a time limit autosave during homework development. The IDE also helped passing out the output information from the compiler, debugger, and student assignment to a safe corner of the repository. The instructions for the dataset goes as an archive folder having three subparts in it as follows:

Source codes – The actual source code assignments submitted by the students are stored in the /src folder inside the archive. The subfolders under ‘src’ are named as A2016, A2017, B2016, and B2017. Each subfolder listed above contains more subfolders inside for individual assignments. Students were required to solve 16-22 assignments in each course, labelled as “Z1/Z1”, “Z1/Z2”, and “Z2/Z1” and so far till the end. The C/C++ source codes solved by the students are stored in these subfolders with an anonymous name. All the traces AutoSaved after every few seconds by the IDE are saved in the archive's stats folder. This folder is again segregated into subfolders named after courses, and the subfolder contains files ending with extension '.stats' for every student (name stays anonymous). The .stats information is stored in JSON format (Key = value pairs). Figure 2 shown below gives a concept map view of the IEEE dataset where there are four courses- A2016, A2017, B2016, B2017 and assignments for each course is described as Z1/Z1..Z5/Z2 for each course.

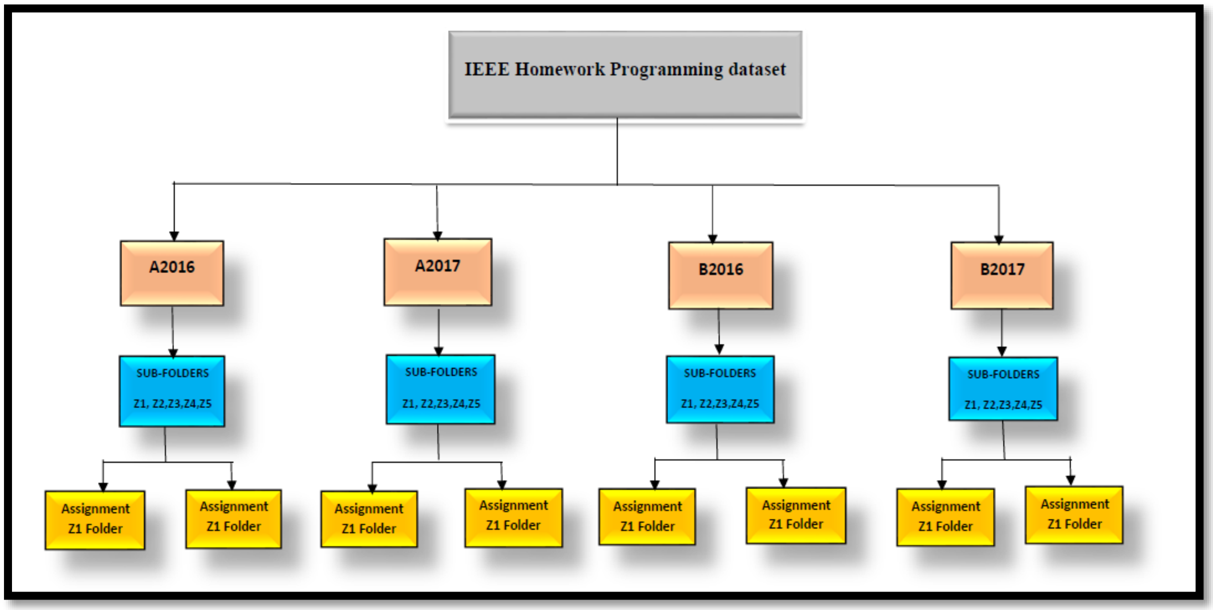
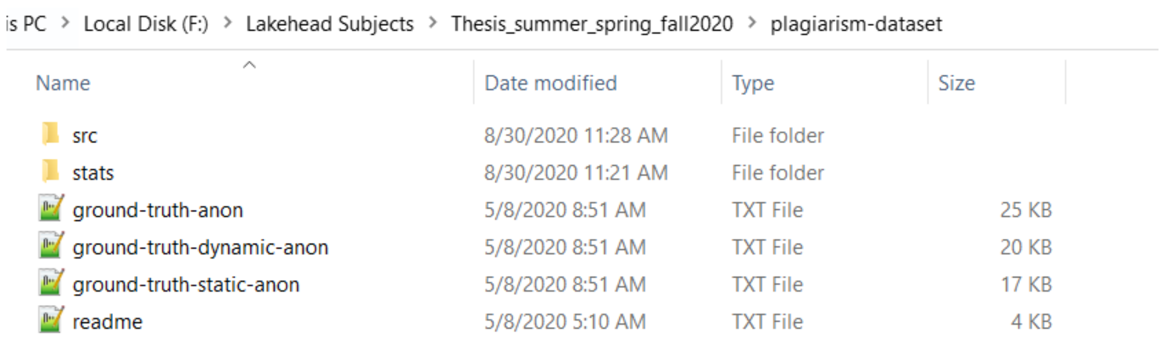


Figure 3: Structure and Composition of IEEE Homework Programming Dataset

Ground Truth- The instructions and format for JSON files is described in the readme.txt file present in the folder as shown in figure 5 below. The ground truth files list the individual and group of students involved in plagiarism due to code similarities detected in their assignments. The three ground truth files starting from 'ground-truth-anon.txt' contain a full list of plagiarisms, ground-truth-static-anon.txt based on source code similarity ground-truth-dynamic-anon.txt based on only failures due to 'oral defence'. Some statistics generated by V. Ljubovic and E. Pajic [24] for the course ‘A’ in 2016 and 2017 i.e. A2016 & A2017 in their latest research published and accepted at IEEE in the year 2020 is shown below in Table 2 as follows:


is PC > Local Disk (F:) > Lakehead Subjects > Thesis_summer_spring_fall2020 > plagiarism-dataset

Name	Date modified	Type	Size
src	8/30/2020 11:28 AM	File folder	
stats	8/30/2020 11:21 AM	File folder	
ground-truth-anon	5/8/2020 8:51 AM	TXT File	25 KB
ground-truth-dynamic-anon	5/8/2020 8:51 AM	TXT File	20 KB
ground-truth-static-anon	5/8/2020 8:51 AM	TXT File	17 KB
readme	5/8/2020 5:10 AM	TXT File	4 KB

Figure 4: Dataset Folder representation

Table 4: Statistics for folder A2016 and A2017

Course	A2016	A2017
Student enrolled	607	488
Number of assignments	18	20
Submitted files	5655	5733
Files per assignment	41-503	125-444
Average file size (bytes)	1567.08	1317.23
Changes per file	1-13821	1-7740
Plagiarized Solutions	746 (13.2%)	699 (12.2%)

Referred to the observations provided in tabl2 1 by the authors [24], we started taking readings and observations for all the C++ source codes based folders, i.e. B2016 and B2017. The courses from ‘B’ consists of all ‘.cpp’ files, and all the assignments subfolders were tested with this proposed system for finding out plagiarized assignment pairs and validate it against the ground truth file. The observations given below in table 3 are the same as from table 2, but associated with folders B2016 and B2017.

Table 5: Statistics for folder B2016 and B2017

Course	B2016	B2017
Student enrolled	607	488
Number of assignments	30	38
Submitted files	12,196	11,192
Files per assignment	41-530	120-300
Average file size (bytes)	6792 Bytes	3692.30 Bytes

We all know that in a three-four-year-long course, the degree of homework participation, in the beginning, is way more than the involvement in the end. If the participation is 90% initially, it closes up to 10-15% in the final semester of the course. As the willing participation increases, the plagiarism decreases, and it’s vice versa in a long-term graduation program. The technique used to overcome the plagiarism index and balance out the proportion was to make 20% of the total students to deliver oral-defense of their homework. The ground truth files were constructed on a marking system where the students who failed to defend their homework defense were marked as 'Plagiarized' in the file. Proper classification of homework is a must needed feature in a similarity detector tool, but every tool handles the situation differently. Some tools have defined a threshold on assignment length. Some have pre-defined heuristics, and a few tools will simply mark all the students as plagiarized and leave unsupervised decisions to a human evaluator. A decent approach for avoiding overfitting with the proposed system in this paper would divide the dataset into training and testing datasets for the underlying machine learning algorithm. As explained at the beginning of this section, the normal ground truth file contains all the plagiarized files. In addition to the normal file, two more ground truth files have been added, such as static for similar homework documents and dynamic ones. They exclude original authors and keep those who have no similar pairs. In the ground file, the assignments are represented in similar files, such as triplets and quadruplets. When it comes to evaluating a newly developed plagiarism tool, one does not need to identify identical document pairs but should count false positives and false negatives inclusive of detected pairs.

For the system under development for this research, the entire assignment folder could be given as an absolute path to the main java program to compare and identify similar assignments. The java program could be tweaked in a possible way of running a big loop by iterating on all the folder assignments of the leading course directory – such as A2016, A2017, B2016 or B2017. This approach has a probability of 90%, resulting in complex challenges like massive processing speed of the CPU/GPU, large storage space and memory and, nonetheless, hours of running time.

2.5 The Software System Compilation Model

As mentioned in the section above, the IEEE homework programming dataset comprises of four folders such as A2016, A2017, B2016 and B2017. Each folder consists of subfolder assignments and each sub-folder assignment further has around 400-500 source code in it. This count is nothing but the number of students taking the particular course ‘A’ or ‘B’ and the course within the same boundary. The assignment subfolder path is fed to the program, and source code files are compared with others for results. As shown in figure 5 below, the source codes are given to the ANTLR (Another tool for Language Recognizer). ANTLR is a powerful Lexer and parser generator and breaks down the ‘C’ and ‘C++’ source codes into tokens such as identifiers, keywords, arithmetic operators, logical operators and other operators. The entire software system using compiler design concepts is explained in three stages or a three-tier system. To give an example of a software engineering design pattern for a system, MVC is quite popular when it comes to web programming or full-stack development. The authors [25] explain the detailed working of a web app and database based on MVC architecture. MVC stands for Model-View-Controller design in which there are three layers defined as follows:

Model – The model layer represents the business layer of the application. The model layer is a set of java classes representing the state of the system at the given time. The model layer has a bidirectional flow from and to View and controller layers.

View – View is the presentation layer where the information processed and store by the model and controller is displayed to the front-end user.

Controller – The controller layer is an interface between Model and View layers. Users can make a fetch or access request from the View layer and the controller fulfills the request. The request has to pass the processing from the model layer before reaching the view layer.

Advantages of MVC architecture:

1. Multiple developers can work with the three layers on individual machines.
2. Scalability and Flexibility to expand and extend the scope of the application
3. Components of the model are less dependent on each other
4. Application following MVC architecture is easy and convenient to understand
5. Software and web testing of an application becomes easy

Refer to figure 5 given below, which shows the exact flow of an MVC system by keeping the user at the center of the system. The controller mostly acts as an interface between model and view and therefore involves of servlets. The proposed system replaces the controller by an interacting database object that triggers the running code at frequent intervals to store the information processed in the relational database at the back-end.

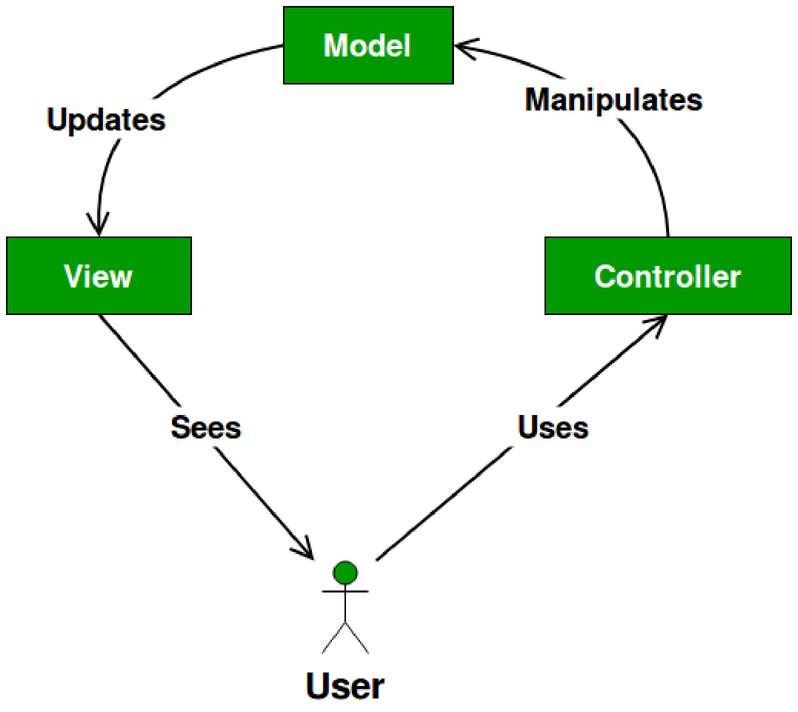


Figure 5: Working of a MVC software Design Framework

The system of interest developed for this thesis follows the MVC design pattern but replacing the controller class by database objects. The authors [26] [27] for code-reusability in small applications invented a lightweight MVC. Our system simply has model- view design architecture where model and controller are located together in one model layer. In other words, our system seems to be running on and following the Model-View architecture as shown in the figure below:

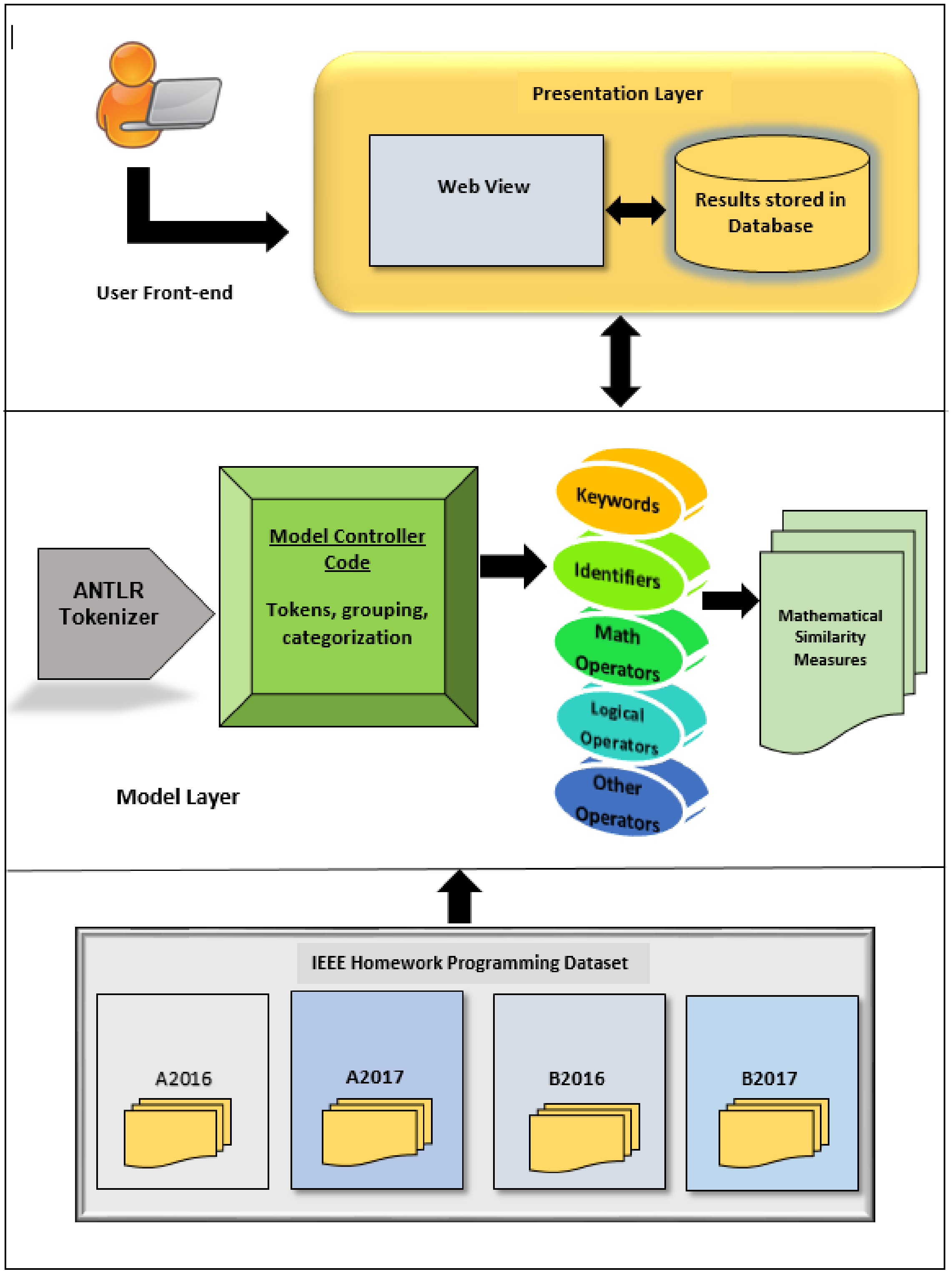


Figure 6: Software System Architecture - Model-Controller and View Architecture

Chapter 3. Forensic Engine Implementation

3.1 ANTLR Tokenizer

The general introduction of ANTLR is given in subsection 1.2.1 methodology of chapter 1. To dive deep into the tool, ANTLR uses a left-to-right, leftmost derivation (LL1) parser for reading and processing textual files. The plugin for ANTLR is available from its website (https://www.antlr.org/) and can be installed in the IDE environment, such as eclipse or IntelliJ IDEA. The ideal IDE platform preferred for developing this kind of system with heavy data handling and building grammars for parsing is Intellij IDEA. This tokenizer in the series is supposed to break down the stream of code into lexemes by referring to the ‘c’ or ‘c++’ grammar. The program has been constructed in such a way that it can detect the extension of source codes in the given path such as ‘.c’ or ‘.cpp’ and choose the grammar file accordingly. The second tokenizer has exclusive use for detecting new lines, comments and line numbers for the corresponding printouts. The lexical tokenizer, which is ANTLR, generates tokens in clusters of identifiers, keywords, arithmetic operators, logical and other operators for both the files and list out the count for each collection, including multiples. The clusters/sets obtained from the source codes are compared with each other based on similarity distance algorithms in the mainframe system. ANTLR can take a piece of text and transform it into AST (Abstract Syntax Tree). We will not be focusing on AST development for the proposed system because of heavy computation and multiple file comparison. ANTLR is strictly used for this system for lexical analysis purposes, where the lexer takes individual characters from the code and transforms them into tokens. A simple example of C/C++ code is broken down into tokens is shown below:

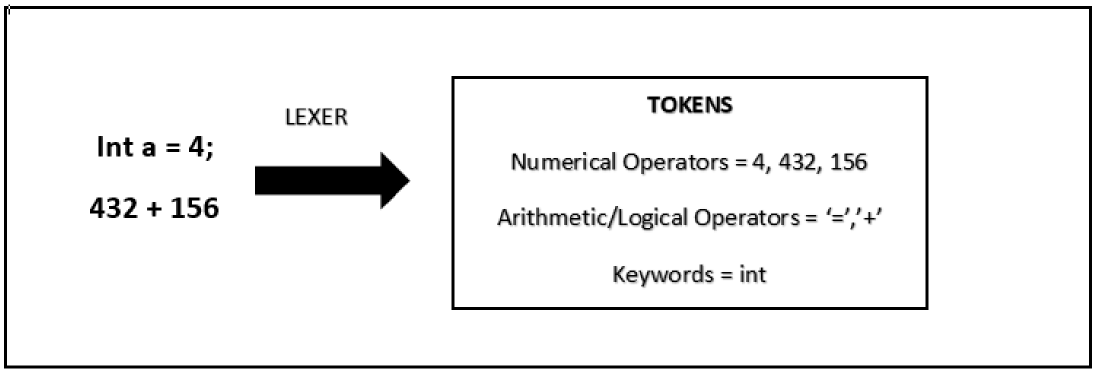

Int a = 4;
432 + 156
LEXER
TOKENS
Numerical Operators = 4, 432, 156
Arithmetic/Logical Operators = ‘=’,’+’
Keywords = int

Figure 7: ANTLR LEXER working

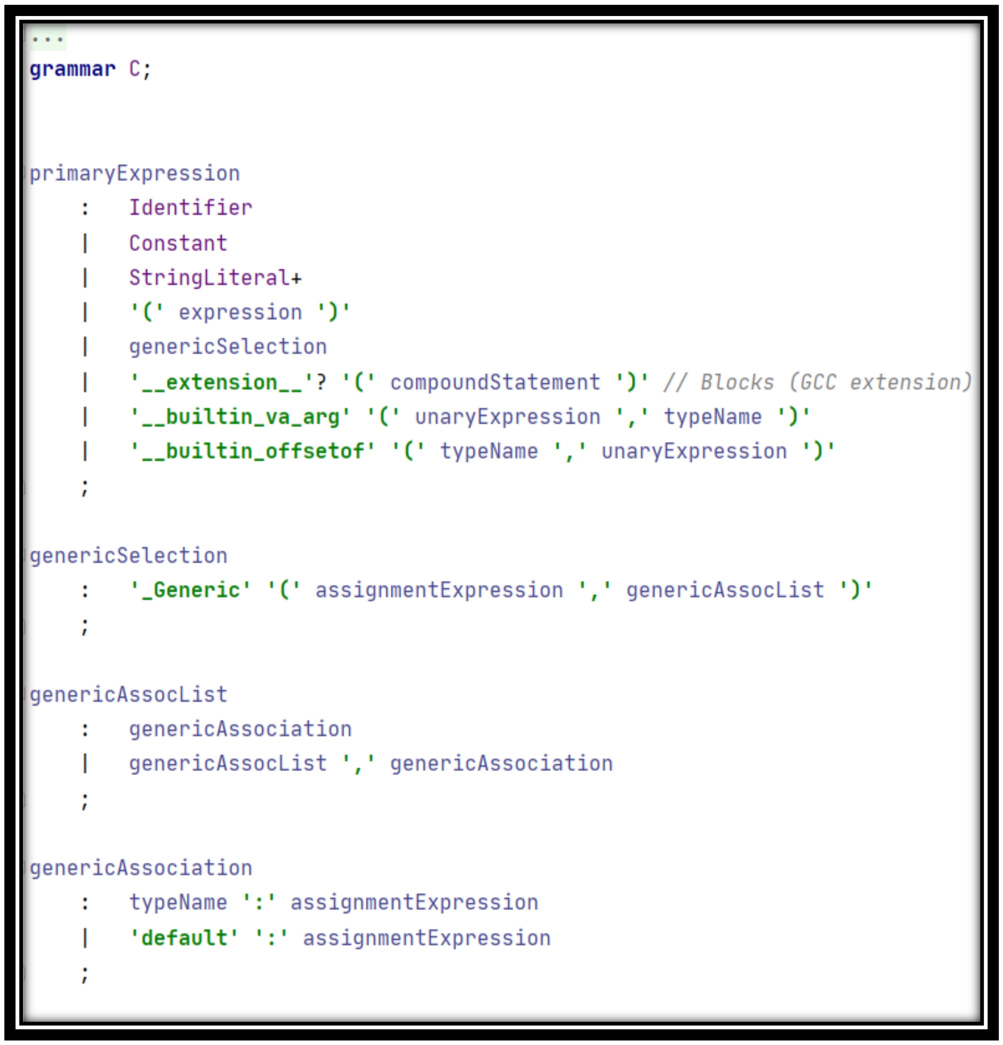
The lexer can only identify the language and separate tokens because we program the lexer to do the task. The code snippet given below is a lexer rule example that tells the program to identify a number and space characters, as shown below:

```
* Lexer Rules
 */
NUMBER     : [0-9]+ ;
WHITESPACE: ' ' -> skip;
```

The complete grammar for C and C++ is embedded with the code as the system is detecting similarities in the same programming language assignments. As we can see that all lexer rules are uppercase, and the parser rules could be lowercase in some cases. They can be ambiguous and are analyzed in the order of appearance. The approach of writing a grammar for any programming language highly depends on the approach of the programming language or the code. A java code can be divided into three sections, such as imports, main structure and type definitions. The basic and preferred approach for writing a java code is the ‘Top-down approach’ where the code's symmetry is retained. The programming approach used for developing our system is the ‘top-down approach’ as java and formatting knowledge are well acquired. The requirements are satisfied from high-level elements to low-level in ascending order. The ‘bottom-up approach’ is only efficient when the main intention is to design a parser because it starts from low-to-high element attention construction.

3.2 C & C++ Grammar

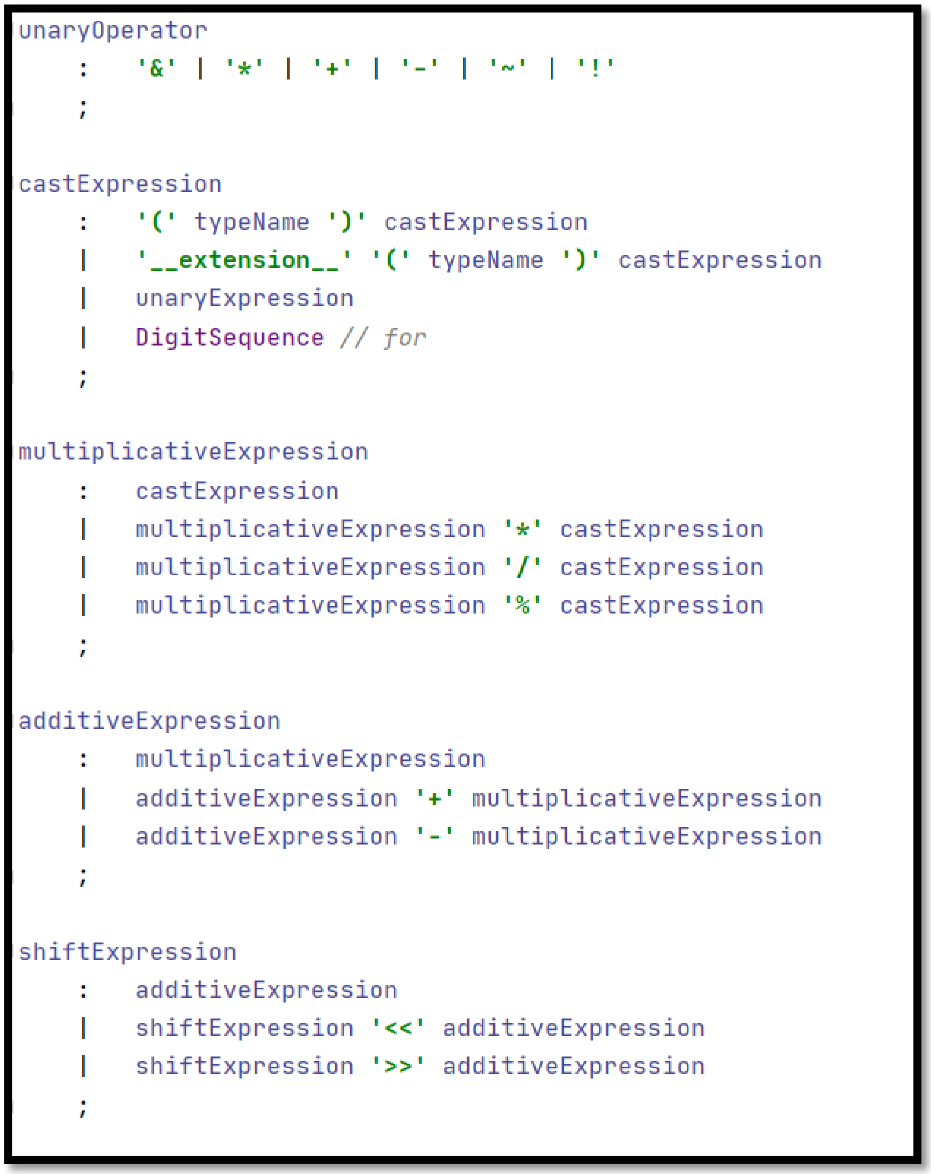
Designing a grammar for C and C++ programming languages is difficult, as it needs to be intuitive for the user and unambiguous to make the user manageable. The initial copy of ‘C’ grammar is available on https://github.com/antlr/grammars-v4/blob/master/c/C.g4 and open for the public to download and do modifications. We will start by defining our grammar for ‘C’ in this paragraph and focus on ‘C++’ later. The description of our ‘C’ grammar designing is as follows:



```
...
grammar C;


primaryExpression
    :   Identifier
    |   Constant
    |   StringLiteral+
    |   '(' expression ')'
    |   genericSelection
    |   '__extension__'? '(' compoundStatement ')' // Blocks (GCC extension)
    |   '__builtin_va_arg' '(' unaryExpression ',' typeName ')'
    |   '__builtin_offsetof' '(' typeName ',' unaryExpression ')'
    ;

genericSelection
    :   '_Generic' '(' assignmentExpression ',' genericAssocList ')'
    ;

genericAssocList
    :   genericAssociation
    |   genericAssocList ',' genericAssociation
    ;

genericAssociation
    :   typeName ':' assignmentExpression
    |   'default' ':' assignmentExpression
    ;
```

Figure 8: C grammar- Defining primary expressions

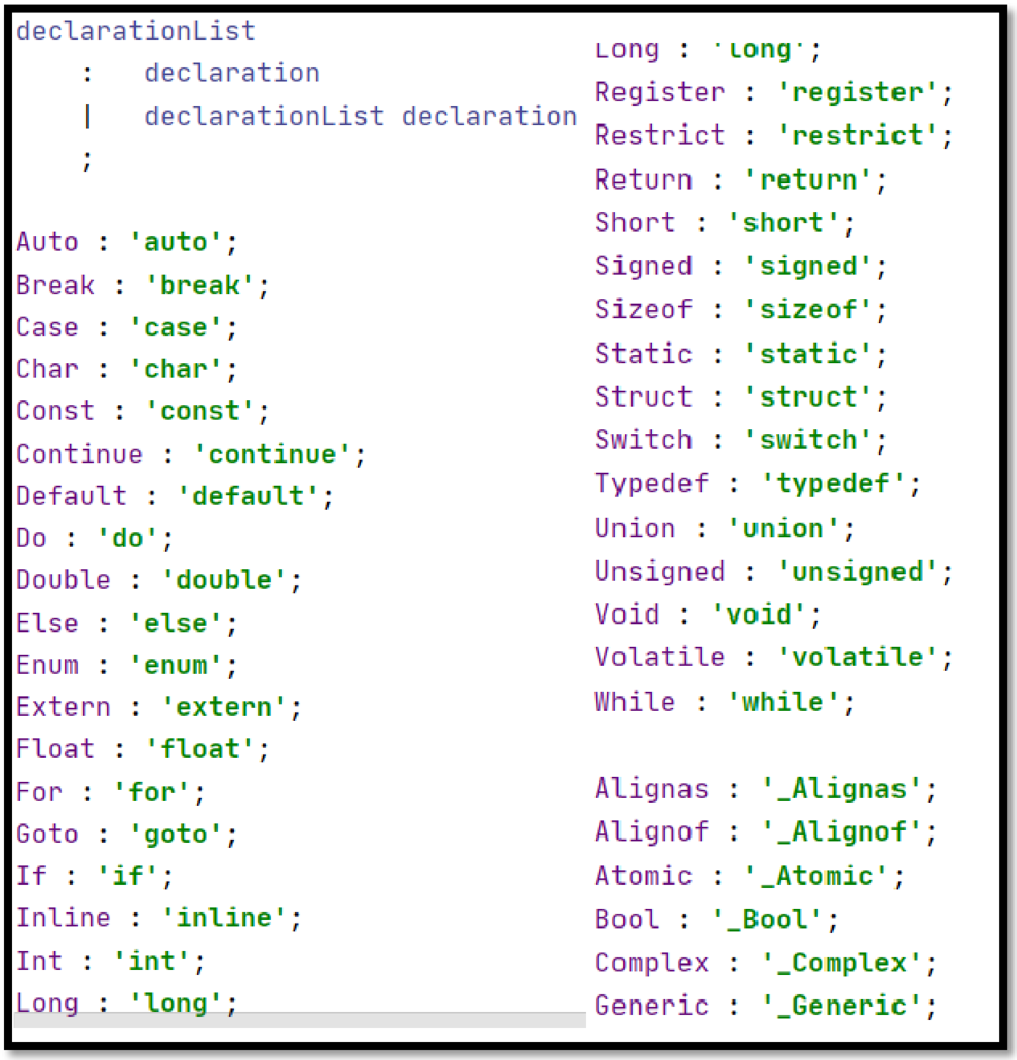
In figure 8, the ‘C’ grammar starts with defining the identifiers, constants and strings under the ‘Primary Expression’ following by generic selection, generic AssocList, and ‘typename’ under genericAssociation.



```
unaryOperator
    :   '&' | '*' | '+' | '-' | '~' | '!'
    ;

castExpression
    :   '(' typeName ')' castExpression
    |   '__extension__' '(' typeName ')' castExpression
    |   unaryExpression
    |   DigitSequence // for
    ;

multiplicativeExpression
    :   castExpression
    |   multiplicativeExpression '*' castExpression
    |   multiplicativeExpression '/' castExpression
    |   multiplicativeExpression '%' castExpression
    ;

additiveExpression
    :   multiplicativeExpression
    |   additiveExpression '+' multiplicativeExpression
    |   additiveExpression '-' multiplicativeExpression
    ;

shiftExpression
    :   additiveExpression
    |   shiftExpression '<<' additiveExpression
    |   shiftExpression '>>' additiveExpression
    ;
```

Figure 9: C grammar- Defining Operator grammar

Referring to figure 9, the unary and binary operators are defined in the C grammar. Not shown in the picture, the other logical operators such as AND, OR, NOR and other operators including brackets (round and square), and ternary operators are defined in the C grammar as well.

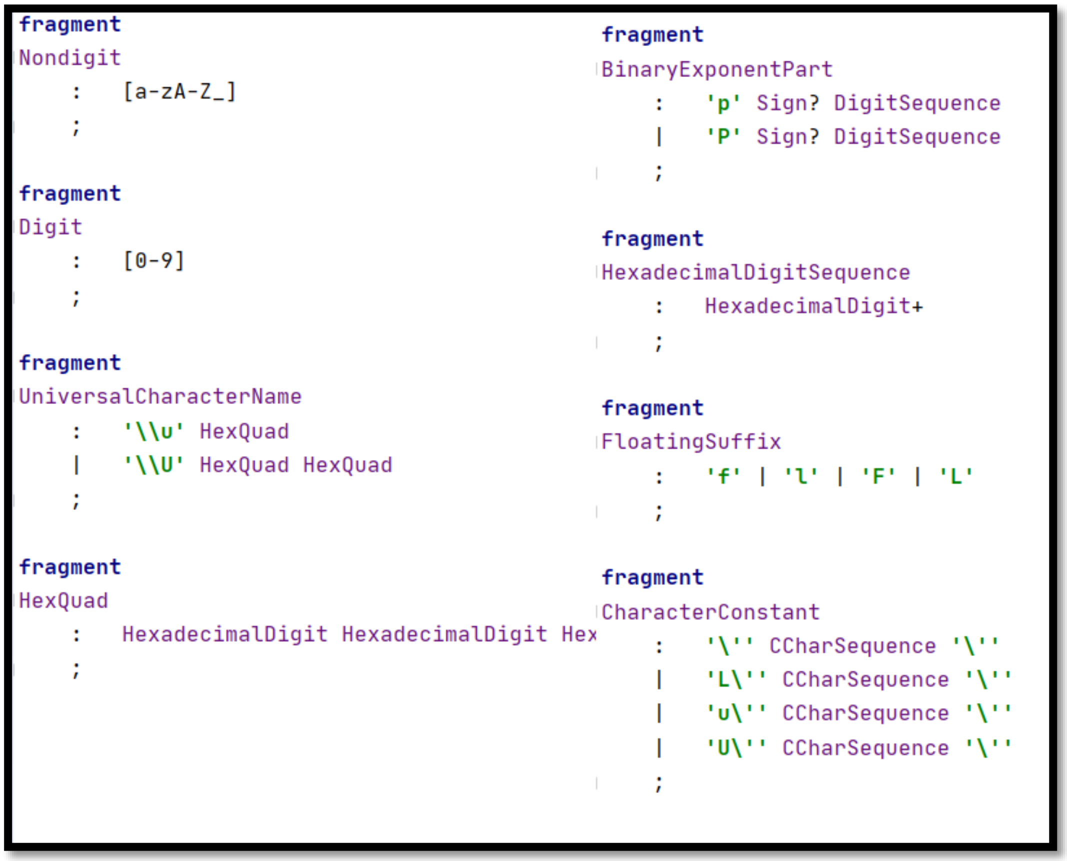


```
declarationList
    :   declaration
    |   declarationList declaration
    ;

Auto : 'auto';
Break : 'break';
Case : 'case';
Char : 'char';
Const : 'const';
Continue : 'continue';
Default : 'default';
Do : 'do';
Double : 'double';
Else : 'else';
Enum : 'enum';
Extern : 'extern';
Float : 'float';
For : 'for';
Goto : 'goto';
If : 'if';
Inline : 'inline';
Int : 'int';
Long : 'long';
```

```
Long : 'long';
Register : 'register';
Restrict : 'restrict';
Return : 'return';
Short : 'short';
Signed : 'signed';
Sizeof : 'sizeof';
Static : 'static';
Struct : 'struct';
Switch : 'switch';
Typedef : 'typedef';
Union : 'union';
Unsigned : 'unsigned';
Void : 'void';
Volatile : 'volatile';
While : 'while';

Alignas : '_Alignas';
Alignof : '_Alignof';
Atomic : '_Atomic';
Bool : '_Bool';
Complex : '_Complex';
Generic : '_Generic';
```

Figure 10: C grammar- Declaring all the keywords and operators

As shown in the figure above, the declaration list declares all the keywords and math operators for C programming Language. The model given below uses rule fragments which are reusable for lexer rules. They are no harm even if defined and not been used in the system. The second half of the figure declares the lexer grammar to skip whitespaces, newlines, line comments and block comments.



```
fragment
Nondigit
    :   [a-zA-Z_]
    ;

fragment
Digit
    :   [0-9]
    ;

fragment
UniversalCharacterName
    :   '\\u' HexQuad
    |   '\\U' HexQuad HexQuad
    ;

fragment
HexQuad
    :   HexadecimalDigit HexadecimalDigit Hex
    ;
```

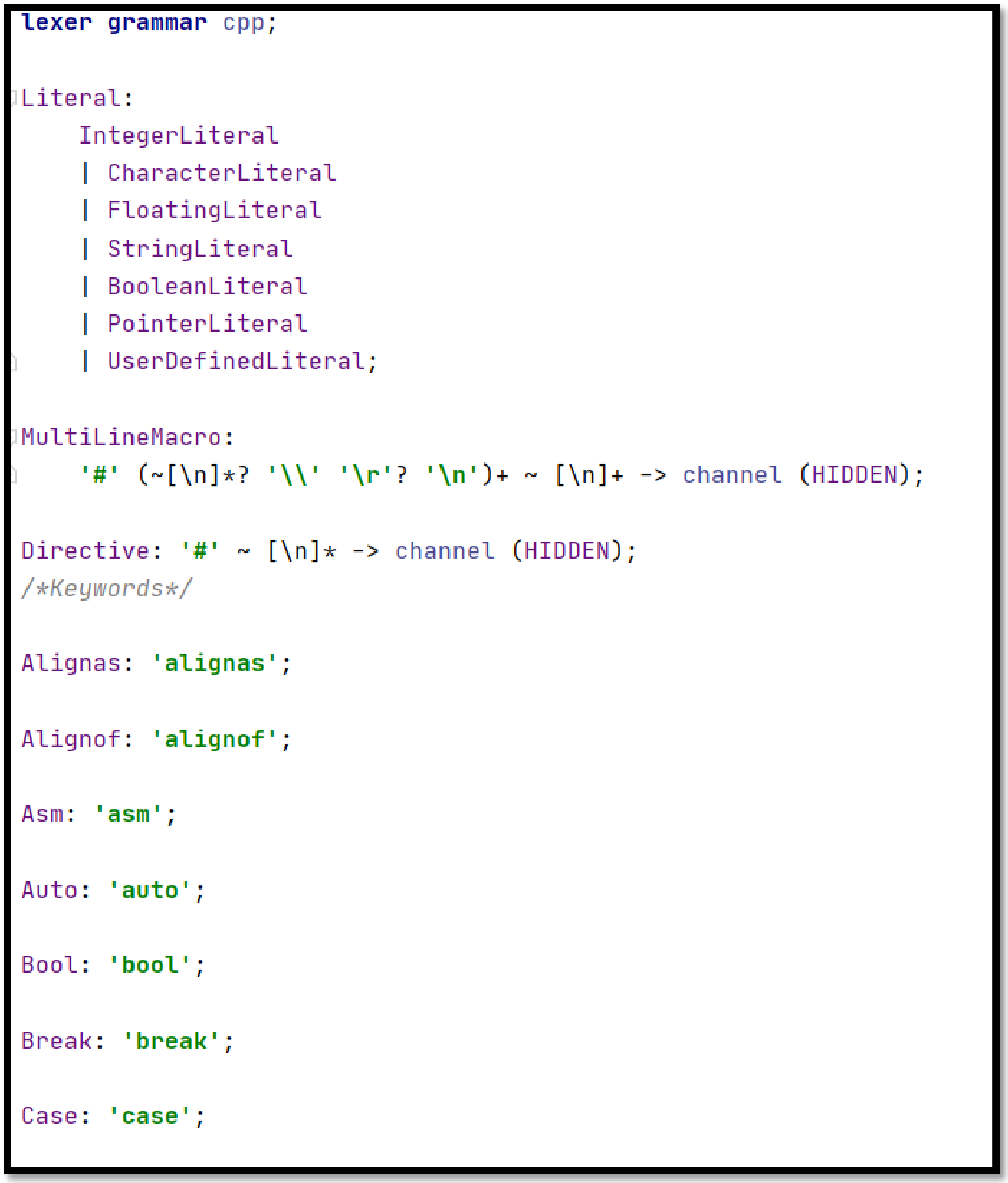
```
fragment
BinaryExponentPart
    :   'p' Sign? DigitSequence
    |   'P' Sign? DigitSequence
    ;

fragment
HexadecimalDigitSequence
    :   HexadecimalDigit+
    ;

fragment
FloatingSuffix
    :   'f' | 'l' | 'F' | 'L'
    ;

fragment
CharacterConstant
    :   '\'' CCharSequence '\''
    |   'L\'' CCharSequence '\''
    |   'u\'' CCharSequence '\''
    |   'U\'' CCharSequence '\''
    ;
```

Figure 11: C Grammar - Defining text fragments and line comments, whitespaces

C++ Grammar Explanation

The grammar for C++ is pretty much the same as C grammar except for a few additional keywords and constants. C++ has the involvement of class methods and functions, and therefore, the grammar is slightly different than that of C programming language. The fragment elements, including the line comments, whitespaces, block comments and fragment constructs, remain the same for this grammar. Below given are the fragments for nonzerodigit, octaldigit, hexadecimaldigit and binarydigit.

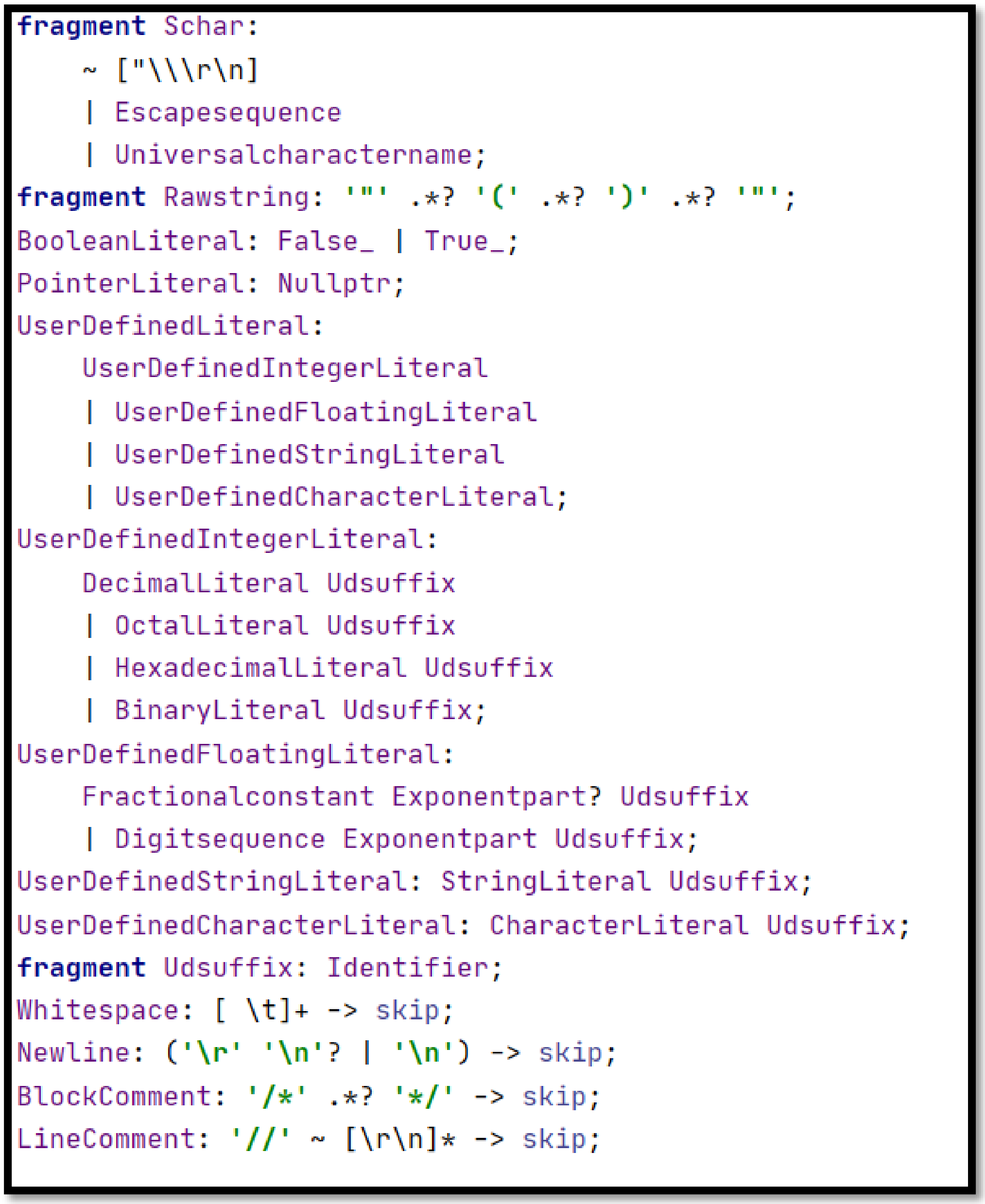
```
fragment NONZERODIGIT: [1-9];

fragment OCTALDIGIT: [0-7];

fragment HEXADECIMALDIGIT: [0-9a-fA-F];

fragment BINARYDIGIT: [01];
```



```
lexer grammar cpp;

Literal:
    IntegerLiteral
    | CharacterLiteral
    | FloatingLiteral
    | StringLiteral
    | BooleanLiteral
    | PointerLiteral
    | UserDefinedLiteral;

MultiLineMacro:
    '#' (~[\n]*? '\\' '\r'? '\n')+ ~ [\n]+ -> channel (HIDDEN);

Directive: '#' ~ [\n]* -> channel (HIDDEN);
/*Keywords*/

Alignas: 'alignas';

Alignof: 'alignof';

Asm: 'asm';

Auto: 'auto';

Bool: 'bool';

Break: 'break';

Case: 'case';
```

Figure 12: C++ Grammar- Defining literals and constants

As shown in the figure above, the literals defined for C++ grammar are integer literal, character literal, floating literal, a string literal, Boolean literal and pointer literal. The additional keywords are declared right after the literals.



```
fragment Schar:
    ~ ["\\\r\n]
    | Escapesequence
    | Universalcharactername;
fragment Rawstring: '"' .*? '(' .*? ')' .*? '"';
BooleanLiteral: False_ | True_;
PointerLiteral: Nullptr;
UserDefinedLiteral:
    UserDefinedIntegerLiteral
    | UserDefinedFloatingLiteral
    | UserDefinedStringLiteral
    | UserDefinedCharacterLiteral;
UserDefinedIntegerLiteral:
    DecimalLiteral Udsuffix
    | OctalLiteral Udsuffix
    | HexadecimalLiteral Udsuffix
    | BinaryLiteral Udsuffix;
UserDefinedFloatingLiteral:
    Fractionalconstant Exponentpart? Udsuffix
    | Digitsequence Exponentpart Udsuffix;
UserDefinedStringLiteral: StringLiteral Udsuffix;
UserDefinedCharacterLiteral: CharacterLiteral Udsuffix;
fragment Udsuffix: Identifier;
Whitespace: [ \t]+ -> skip;
Newline: ('\r' '\n'? | '\n') -> skip;
BlockComment: '/*' .*? '*/' -> skip;
LineComment: '//' ~ [\r\n]* -> skip;
```

Figure 13: C++ Grammar- Defining fragments

Figure 14 shows the declaration of literals with respect to the suffix and fragments. The fragment elements declared include whitespaces, newline and block comment.

3.3 First Phase – Lexical Analysis

Lexical analysis is defined as the first phase of the compiler in which the source code is analyzed and broken down into tokens, of which sentences are formed. The lexer or the lexical analyzer could be modified so that the whitespaces block comments and comments could be removed from the source code.

Finding invalid tokens highly depends on the configuration of the tokenizer. Let us say there is a bag of words that is organized in the lexer's spectrum (code). The tokenizer will validate the source code against that bag of words, where the matching words will be accepted as valid tokens and given as output/ store according to the user convenience. Adding to the acceptance of valid tokens, some pre-defined rules for every lexeme are identified as valid. Grammar rules define these rules with the help of patterns. A pattern is a mixed entity of regular expressions and is used to define a token. As shown in the above given figure 7, ANTLR is used as a tokenizer in this system and accepts valid tokens in form of identifiers, keywords, math operators, logical operators and other operators besides math and logic.

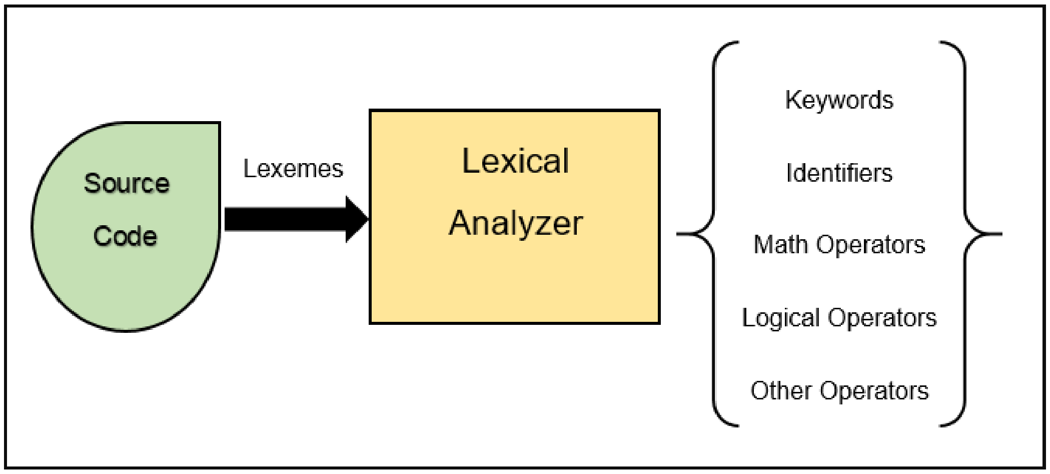


Figure 14: Tokenization: Lexical Analysis

Specifications of Tokens:

Alphabets: Any finite set of {0, 1} symbols is a set of binary alphabets and {A, B, C, D, E, F to Z} is a set of hexadecimal alphabets. [a-z, A-Z] is a set of English language alphabets.

Strings: A finite set of alphabets from the above-mentioned alphabets is called a string. For the given lexer, the examples of strings can be given as ‘hello’, ‘world’, ‘print’ and etc.

Special symbols

The special symbols from a specific high-level language are mentioned in the table given below:

Table 6: Symbol Table Specification

Arithmetic operators	Addition(+), subtraction(-), multiplication(*), division(/), modulus (%),
Logical operators	And (&&), Or (||), Not (!)
Relational operators	Less than(<), less than equal (<=), more than(>), more than equal(>=), equal equal (==), not equal (!=)
Other Operators	! , comma(,), semicolon(;), dot(.), arrow(->)
Location specifier	&
Assignment	=
Shift Operator	>>, >>>, <<, <<<
Preprocessor	#

3.3.1 Tokenizing the source codes

In this subsection, we will discuss the procedure of tokenizing the source code, i.e. breaking down the source code into tokens or lexemes. The overview for lexical analysis using ANTLR is given in the above sections, but the system implementation for ‘SimDec’ forensic engine stage 1: Lexical analysis will be presented in this subsection, as shown below for a source code example of ‘C’ and ‘C++.’

Table 7: Tokenization Implementation for 'C' file

```
#include <stdio.h>
#include <math.h>
#define epsilon 0.000001
int main() {
      float a1,a2,b1,b2,x,y;
      printf("Unesite a1,b1,a2,b2: ");
      scanf("%f, %f, %f, %f",&a1,&b1,&a2,&b2);
      x=(b2-b1)/(a1-a2);
      y=a1*x+b1;
      if(fabs(a1-a2)<epsilon && fabs(b1-b2)>epsilon)
printf("Paralelne su");
      else if( fabs(a1-a2)<epsilon && fabs(b1-b2)<epsilon)
printf("Poklapaju se");
      else printf("Prave se sijeku u tacci (%.1f,%.1f)",x,y);


      return 0;
}
```

The ‘C’ source file – ‘student4959.c’ shown above is tokenized or put through lexical analysis in the proposed system, and then tokens/lexemes obtained are given below as follows:

```
"C:\Program Files\Java\jdk-14.0.1\bin\java.exe" "-
javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition
2020.2.1\lib\idea_rt.jar=61954:C:\Program Files\JetBrains\IntelliJ
IDEA Community Edition 2020.2.1\bin" -Dfile.encoding=UTF-8 -
classpath
C:\Users\batma\IdeaProjects\demoHello\out\production\demoHello;C:\Us
ers\batma\Downloads\antlrjar\antlr-4.8-
complete.jar;C:\Users\batma\Downloads\commons-io-
2.7.jar;C:\Users\batma\Downloads\commons-text-
1.8.jar;C:\Users\batma\Downloads\mysql-connector-java-8.0.20\mysql-
connector-java-8.0.20\mysql-connector-java-8.0.20.jar
com.company.com.company.Runner
```

Table 8: Tokens of 'C' source code

epsilon	0.000001	int	main	(
)	{	float	a1	,
a2	,	b1	,	b2
,	x	,	y	;
printf	(	"Unesite a1,b1,a2,b2: "	)	;
scanf	(	"%f, %f, %f, %f"	,	&
a1	,	&	b1	,
&	a2	,	&	b2
)	;	x	=	(
b2	-	b1	)	/

(	a1	-	a2	)
;	y	=	a1	*
x	+	b1	;	if
(	fabs	(	a1	-
a2	)	<	epsilon	&&
fabs	(	b1	-	b2
)	>	epsilon	)	printf
(	"Paralelne su"	)	;	else
if	(	fabs	(	a1
-	a2	)	<	epsilon
&&	fabs	(	b1	-
b2	)	<	epsilon	)
printf	(	"Poklapaju se"	)	;
else	printf	(	"Prave se sijeku u tacci (%.1f,%.1f)"	,
x	,	y	)	;
return	0	;	}	

The lexical analysis identifies a string text as a complete string in between the double-quotes. For example, print statement printing text in double-quotes. According to our context, the ANTLR tokenizer is configured in particular way and, therefore, difficult to alter the default configurations. For example, the word "Prave se sijeku u tacci (%.1f,%.1f)" has operators in it, but being enclosed in double-quotes, the entire string is detected as an 'Identifier'. This cannot necessarily be seen as a disadvantage because the detection strength is not weakened due to this one flaw.

Going ahead and executing the set of commands for grouping the tokens into categories for 'C' source codes by referring the 'C' grammar is represented as follows:

```
==============================================

student4959.c

student1326.c

==============================================

The Lexemes for file1: [0.000001, int, main, (, ), {, float, a1, ,, a2, ,, b1, ,, b2, ,, x, ,, y, ;, printf, (, "Unesite a1,b1,a2,b2: ", ), ;, scanf, (, "%f, %f, %f, %f", ,, &, a1, ,, &, b1, ,, &, a2, ,, &, b2, ), ;, x, =, (, b2, -, b1, ), /, (, a1, -, a2, ), ;, y, =, a1, *, x, +, b1, ;, if, (, fabs, (, a1, -, a2, ), <, epsilon, &&, fabs, (, b1, -, b2, ), >, epsilon, ), printf, (, "Paralelne su", ), ;, else, if, (,
```

fabs, (, a1, -, a2,), <, epsilon, &&, fabs, (, b1, -, b2,), <, epsilon,), printf, (, "Poklapaju se",), ;, else, printf, (, "Prave se sijeku u tacci (%.1f,%.1f)", ,, x, ,, y,), ;, return, 0, ;, }, <EOF>]

129

Others: [(,), {, (,), (,), (,), (,), (, (,), (,),), (,), (, (,), (,),), (,), (,), }]

Numerical Values: [0.000001, 0]

Keywords: [int, float, if, if, else, else, return]

Logical Operators: [<, <, <, >, &&, &&, &, &, &, &]

Math Operators: [=, =, -, -, -, -, -, -, +, *, /]

Identifiers: [main, a1, a2, b1, b2, x, y, printf, scanf, fabs, epsilon, "Paralelne su", "Poklapaju se"]

The java codes for grouping of lexemes in the 'tokenizer customization' phase have been described in the appendix for java programming at the end of the thesis. The 'C++' source file – 'student1044.cpp' shown below is tokenized or put through lexical analysis in the proposed system, and then tokens/lexemes obtained are given below as follows

```
    NAPOMENA: i javni ATo-vi su dio postavke

    Autotestovi by Berina Cocalic. Sva pitanja, sugestije
    i prijave gresaka saljite na mail: bcocalic1@etf.unsa.ba

*/
#include<iostream>
#include<vector>
#include<string>
using namespace std;

        void IzbaciDuple (vector<string>&v){


            for(int i(0);i<int(v.size());i++){
            for(int j(i+1);j<int(v.size());j++){
                if(v[i]==v[j]){
                v.erase(v.begin()+j);
                j--;
                }

            }
        }
```

```
int main ()
{
      vector<string>v{"Ja","se","Lejla", "Lejla","se", "zovem", 
"Lejla", "Lejla"};
      IzbaciDuple(v);
      for(int i(0);i<v.size();i++)
      cout<<v[i];
      return 0;
}
```

The tokens given out after processing the 'C++' source code file are large and, therefore, difficult to accumulate in a table for display. The cut-short version of all the lexemes is shown below as follows:

```
"C:\Program Files\Java\jdk-14.0.1\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ 
IDEA Community Edition 2020.2.1\lib\idea_rt.jar=53361:C:\Program Files\JetBrains\IntelliJ IDEA 
Community Edition 2020.2.1\bin" -Dfile.encoding=UTF-8 -classpath 
C:\Users\batma\IdeaProjects\demoHello\out\production\demoHello;C:\Users\batma\Downloads\a
ntlrjar\antlr-4.8-complete.jar;C:\Users\batma\Downloads\commons-io-
2.7.jar;C:\Users\batma\Downloads\commons-text-1.8.jar;C:\Users\batma\Downloads\mysql-
connector-java-8.0.20\mysql-connector-java-8.0.20\mysql-connector-java-8.0.20.jar 
com.company.com.company.Runcpp
```

Table 9: Tokenization of a 'C++' source code file

#include <iostream>	#include <vector>	#include <deque>	#include <iterator>	#include <algorithm>	#include <iomanip>	using
std	::	cout	;	using	std	::
cin	;	int	suma	(	int	n
)	{	int	sumacif	=	0	;
while	(	n	!=	0	)	{
sumacif	+=	(	n	%	10	)
;	n	/=	10	;	}	return
sumacif	;	}	int	SumaDjelilaca	(	long
long	int	x	)	{	int	suma
(	0	)	;	if	(	x
<	0	)	x	=	-	x
;	for	(	int	i	=	1
;	i	<=	x	;	i	++
)	{	if	(	x	%	i
==	0	)	suma	+=	i	;
}	return	suma	;	}	bool	Prost
(	int	x	)	{	for	(
int	i	=	2	;	i	<
x	;	i	++	)	{	if

(	x	%	i	==	0	)
return	false	;	}	return	true	;
}	int	BrojProstihFaktora	(	long	long	int
x	)	{	int	br	(	0
)	;	if	(	x	<	0
)	x	=	-	x	;	for
(	int	i	=	2	;	i
<	x	;	i	++	)	{
if	(	x	%	i	==	

The lexeme grouping for C++ source code file into categories such as others, numerical values, logical operators, math operators and identifiers is shown as follows:

```
=============================================

student1044.cpp

student1029.cpp

=============================================

Others: [(, ), {, (, (, ), (, (, ), ), ), {, (, (, ), (, (, ), ), ),
{, (, [, ], [, ], ), {, (, (, ), ), }, }, }, }, (, ), {, {, }, (, ),
(, (, ), (, ), ), [, ], }]

Numerical Values: [0, 1, 0, 0]

Keywords: [int, int, int, int, int, int, if, return, for, for, for,
void, using, namespace]

Logical Operators: [<, <, <, <, <, >, >, <<, &]

Math Operators: [+, +, ++, ++, ++, --]

Identifiers: [std, IzbaciDuple, vector, string, v, i, size, j,
erase, begin, main, "Ja", "se", "Lejla", "zovem", cout]
```

The two files under comparison go through the lexical analysis process, and tokenization for both the files takes place before the later phases. In the next section, the details regarding the implementation of string similarity or distance similarity algorithms have been illustrated.

3.4 Second Phase – Computations

This is the second phase of our ‘SimDec’ forensic engine. The string similarity techniques' actual mathematical calculations are executed on the data extracted from the ‘C’ or ‘C++’ source codes. The distance & string similarity algorithms chosen for this experimentation are as follows:

Levenshtein Distance Measure

Jaro Distance Measure

Jaro-Winkler Distance Measure

Cosine Similarity

Dice Coefficient

Least Common Substring (LCB)

All the similarity measures mentioned above have been applied to the data extracted from the source codes such as keywords, math operators, logical operators, other operators and numerical are mentioned in chapter 4 of this book.

3.4.1 Execution of Distance, Token, Sequence Similarity Algorithms

Several kinds of similarity algorithms, such as edit distance, token-based and sequence algorithms, are used in defining the statistical model of ‘SimDec’ forensic engine. The figure below describes how, where and when the similarity algorithms are applied to the data extracted from the C/C++ source code corpus. When we say extracting data from the corpus, comparison is information obtained from lexical analysis. The information primarily being in form of tokens / set of tokens such as sets of keywords (file 1, file 2 in the figure), sets of identifiers, sets of numerical values and other as illustrated in figure 15.

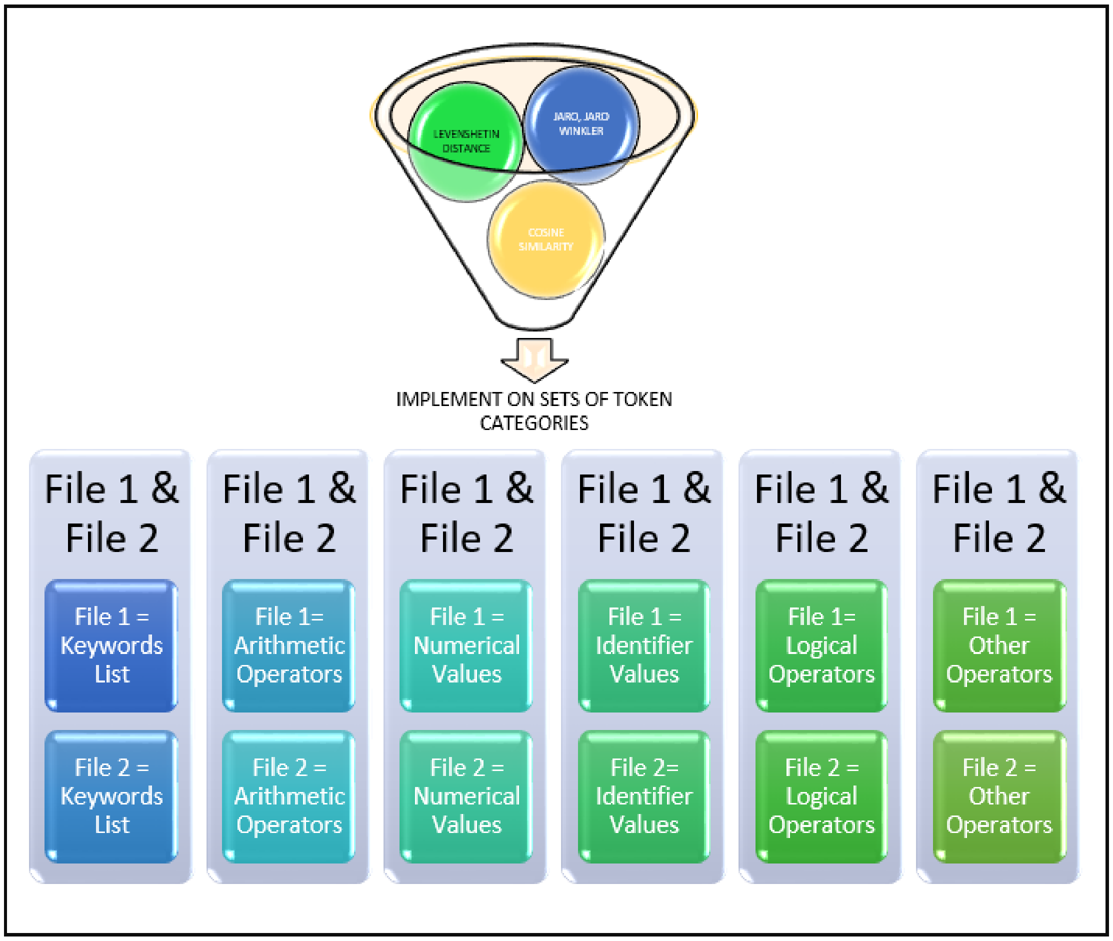
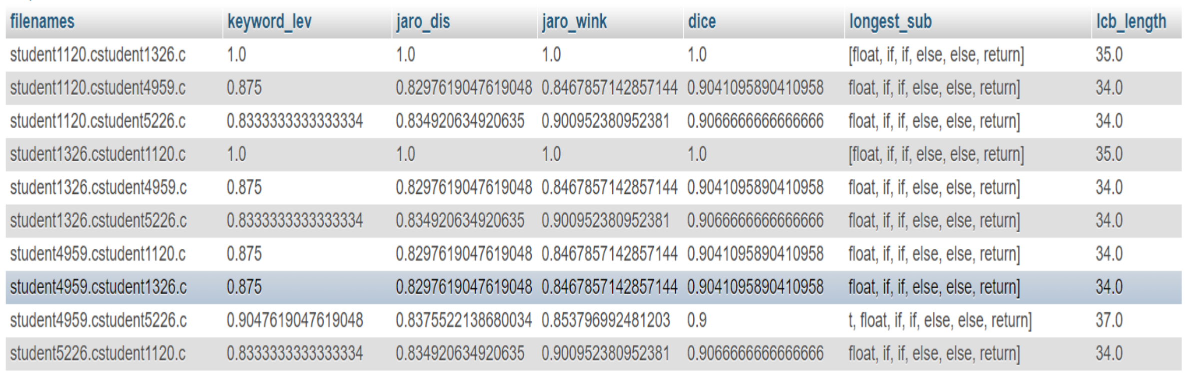

LEVENSHETIN DISTANCE
JARO, JARO WINKLER
COSINE SIMILARITY
IMPLEMENT ON SETS OF TOKEN CATEGORIES
File 1 & File 2
File 1 = Keywords List
File 2 = Keywords List
File 1 & File 2
File 1= Arithmetic Operators
File 2 = Arithmetic Operators
File 1 & File 2
File 1 = Numerical Values
File 2 = Numerical Values
File 1 & File 2
File 1 = Identifier Values
File 2= Identifier Values
File 1 & File 2
File 1= Logical Operators
File 2 = Logical Operators
File 1 & File 2
File 1 = Other Operators
File 2 = Other Operators

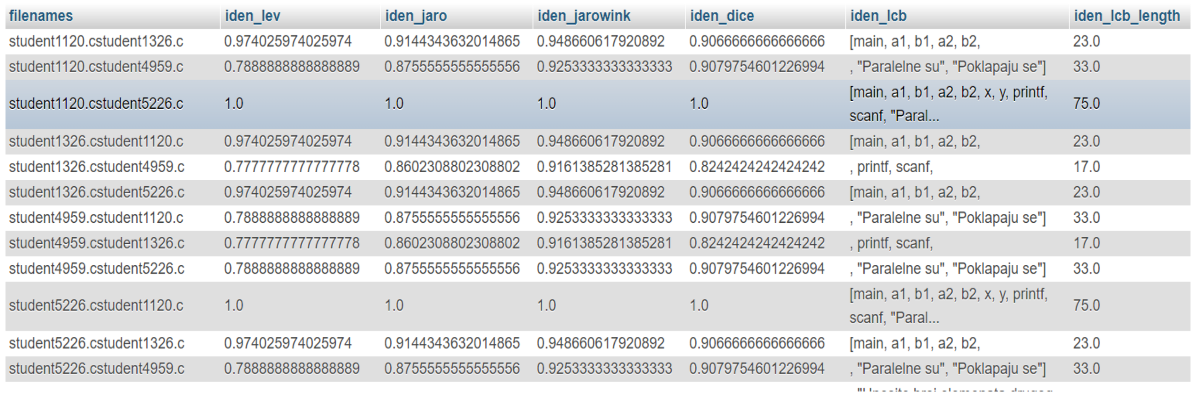
Figure 15: Execution of Similarity techniques on extracted data

Here, the techniques used for similarity measures are cosine similarity, Levenshtein distance, and Jaro-Jaro Winkler and dice coefficients. In the SimDec engine, two files from a folder are under evaluation at the moment (in a loop) and do the same for other files once it exits the loop after evaluating two files in a queue. The information for keywords score, identifier score, arithmetic operators score and others is stored in different MySQL tables of the same database at first. The scores are further aggregated as one whole number, and the final computation takes following the first. Let's review all the table singles for each token category starting with keywords score, as shown below:



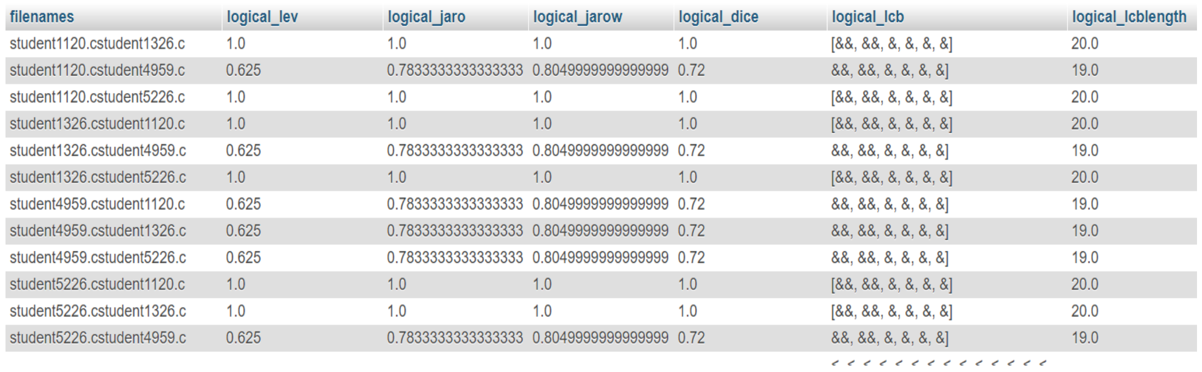
filenames	keyword_lev	jaro_dis	jaro_wink	dice	longest_sub	lcb_length
student1120.cstudent1326.c	1.0	1.0	1.0	1.0	[float, if, if, else, else, return]	35.0
student1120.cstudent4959.c	0.875	0.8297619047619048	0.8467857142857144	0.9041095890410958	float, if, if, else, else, return]	34.0
student1120.cstudent5226.c	0.8333333333333334	0.834920634920635	0.900952380952381	0.9066666666666666	float, if, if, else, else, return]	34.0
student1326.cstudent1120.c	1.0	1.0	1.0	1.0	[float, if, if, else, else, return]	35.0
student1326.cstudent4959.c	0.875	0.8297619047619048	0.8467857142857144	0.9041095890410958	float, if, if, else, else, return]	34.0
student1326.cstudent5226.c	0.8333333333333334	0.834920634920635	0.900952380952381	0.9066666666666666	float, if, if, else, else, return]	34.0
student4959.cstudent1120.c	0.875	0.8297619047619048	0.8467857142857144	0.9041095890410958	float, if, if, else, else, return]	34.0
student4959.cstudent1326.c	0.875	0.8297619047619048	0.8467857142857144	0.9041095890410958	float, if, if, else, else, return]	34.0
student4959.cstudent5226.c	0.9047619047619048	0.8375522138680034	0.853796992481203	0.9	t, float, if, if, else, else, return]	37.0
student5226.cstudent1120.c	0.8333333333333334	0.834920634920635	0.900952380952381	0.9066666666666666	float, if, if, else, else, return]	34.0

Figure 16: Screenshot of ‘Keywords score’ table with Similarity Techniques



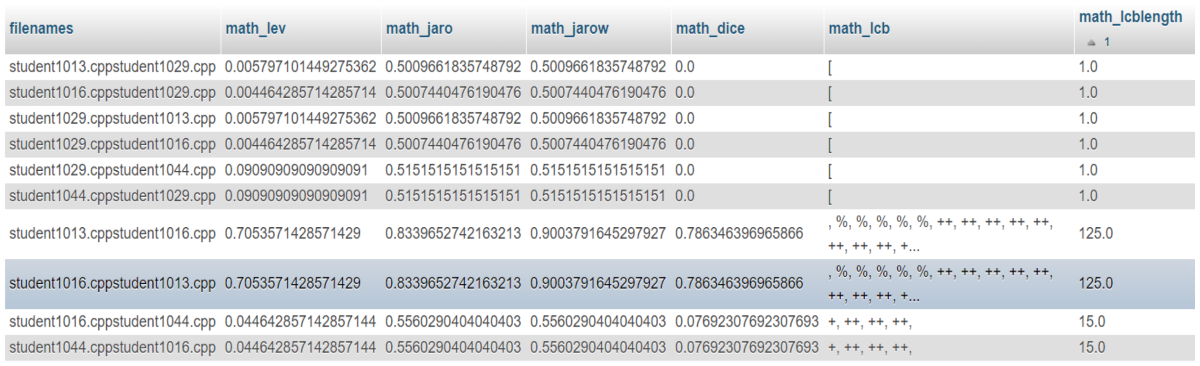
filenames	iden_lev	iden_jaro	iden_jarowink	iden_dice	iden_lcb	iden_lcb_length
student1120.cstudent1326.c	0.974025974025974	0.9144343632014865	0.948660617920892	0.9066666666666666	[main, a1, b1, a2, b2,	23.0
student1120.cstudent4959.c	0.7888888888888889	0.8755555555555556	0.9253333333333333	0.9079754601226994	, "Paralelne su", "Poklapaju se"]	33.0
student1120.cstudent5226.c	1.0	1.0	1.0	1.0	[main, a1, b1, a2, b2, x, y, printf, scanf, "Paral...	75.0
student1326.cstudent1120.c	0.974025974025974	0.9144343632014865	0.948660617920892	0.9066666666666666	[main, a1, b1, a2, b2,	23.0
student1326.cstudent4959.c	0.7777777777777778	0.8602308802308802	0.9161385281385281	0.8242424242424242	, printf, scanf,	17.0
student1326.cstudent5226.c	0.974025974025974	0.9144343632014865	0.948660617920892	0.9066666666666666	[main, a1, b1, a2, b2,	23.0
student4959.cstudent1120.c	0.7888888888888889	0.8755555555555556	0.9253333333333333	0.9079754601226994	, "Paralelne su", "Poklapaju se"]	33.0
student4959.cstudent1326.c	0.7777777777777778	0.8602308802308802	0.9161385281385281	0.8242424242424242	, printf, scanf,	17.0
student4959.cstudent5226.c	0.7888888888888889	0.8755555555555556	0.9253333333333333	0.9079754601226994	, "Paralelne su", "Poklapaju se"]	33.0
student5226.cstudent1120.c	1.0	1.0	1.0	1.0	[main, a1, b1, a2, b2, x, y, printf, scanf, "Paral...	75.0
student5226.cstudent1326.c	0.974025974025974	0.9144343632014865	0.948660617920892	0.9066666666666666	[main, a1, b1, a2, b2,	23.0
student5226.cstudent4959.c	0.7888888888888889	0.8755555555555556	0.9253333333333333	0.9079754601226994	, "Paralelne su", "Poklapaju se"]	33.0

Figure 17: Screenshot of ‘Identifiers score’ table with Similarity Techniques



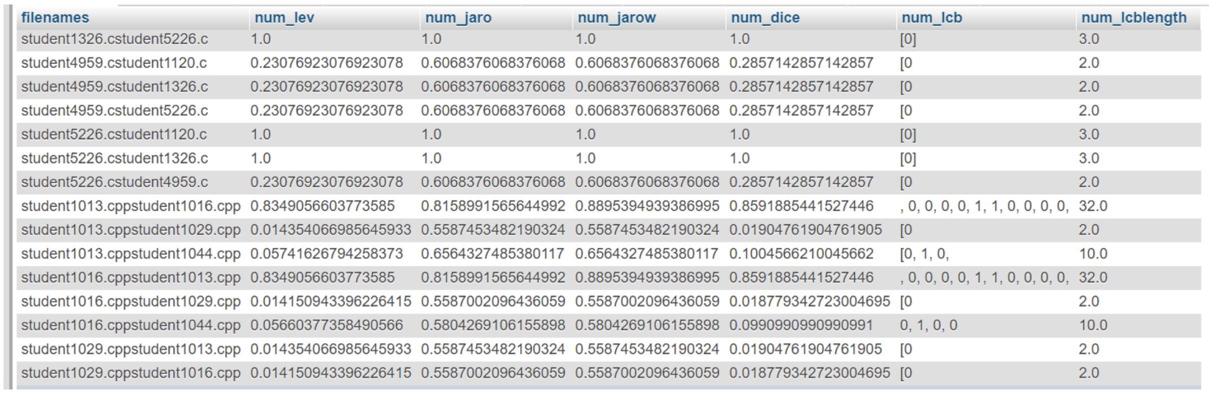
filenames	logical_lev	logical_jaro	logical_jarow	logical_dice	logical_lcb	logical_lcblength
student1120.cstudent1326.c	1.0	1.0	1.0	1.0	[&&, &&, &, &, &, &]	20.0
student1120.cstudent4959.c	0.625	0.7833333333333333	0.8049999999999999	0.72	&&, &&, &, &, &, &]	19.0
student1120.cstudent5226.c	1.0	1.0	1.0	1.0	[&&, &&, &, &, &, &]	20.0
student1326.cstudent1120.c	1.0	1.0	1.0	1.0	[&&, &&, &, &, &, &]	20.0
student1326.cstudent4959.c	0.625	0.7833333333333333	0.8049999999999999	0.72	&&, &&, &, &, &, &]	19.0
student1326.cstudent5226.c	1.0	1.0	1.0	1.0	[&&, &&, &, &, &, &]	20.0
student4959.cstudent1120.c	0.625	0.7833333333333333	0.8049999999999999	0.72	&&, &&, &, &, &, &]	19.0
student4959.cstudent1326.c	0.625	0.7833333333333333	0.8049999999999999	0.72	&&, &&, &, &, &, &]	19.0
student4959.cstudent5226.c	0.625	0.7833333333333333	0.8049999999999999	0.72	&&, &&, &, &, &, &]	19.0
student5226.cstudent1120.c	1.0	1.0	1.0	1.0	[&&, &&, &, &, &, &]	20.0
student5226.cstudent1326.c	1.0	1.0	1.0	1.0	[&&, &&, &, &, &, &]	20.0
student5226.cstudent4959.c	0.625	0.7833333333333333	0.8049999999999999	0.72	&&, &&, &, &, &, &]	19.0

Figure 18: Screenshot of ‘Logical score’ table with Similarity Techniques



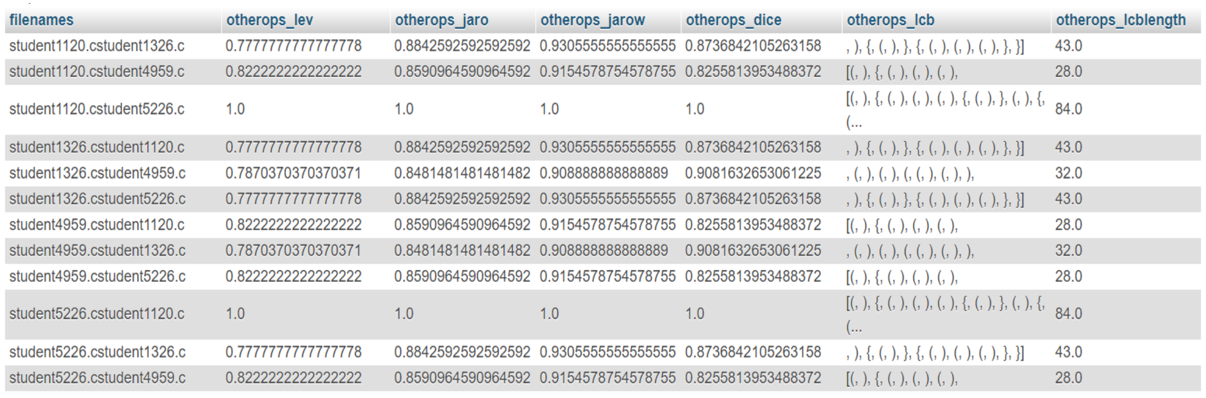
filenames	math_lev	math_jaro	math_jarow	math_dice	math_lcb	math_lcblength ▲ 1
student1013.cppstudent1029.cpp	0.005797101449275362	0.5009661835748792	0.5009661835748792	0.0	[	1.0
student1016.cppstudent1029.cpp	0.004464285714285714	0.5007440476190476	0.5007440476190476	0.0	[	1.0
student1029.cppstudent1013.cpp	0.005797101449275362	0.5009661835748792	0.5009661835748792	0.0	[	1.0
student1029.cppstudent1016.cpp	0.004464285714285714	0.5007440476190476	0.5007440476190476	0.0	[	1.0
student1029.cppstudent1044.cpp	0.09090909090909091	0.5151515151515151	0.5151515151515151	0.0	[	1.0
student1044.cppstudent1029.cpp	0.09090909090909091	0.5151515151515151	0.5151515151515151	0.0	[	1.0
student1013.cppstudent1016.cpp	0.7053571428571429	0.8339652742163213	0.9003791645297927	0.786346396965866	, %, %, %, %, %, ++, ++, ++, ++, ++, ++, ++, ++, +...	125.0
student1016.cppstudent1013.cpp	0.7053571428571429	0.8339652742163213	0.9003791645297927	0.786346396965866	, %, %, %, %, %, ++, ++, ++, ++, ++, ++, ++, ++, +...	125.0
student1016.cppstudent1044.cpp	0.044642857142857144	0.5560290404040403	0.5560290404040403	0.07692307692307693	+, ++, ++, ++,	15.0
student1044.cppstudent1016.cpp	0.044642857142857144	0.5560290404040403	0.5560290404040403	0.07692307692307693	+, ++, ++, ++,	15.0

Figure 19: Screenshot of 'Math Score' table with Similarity Techniques



filenames	num_lev	num_jaro	num_jarow	num_dice	num_lcb	num_lcblength
student1326.cstudent5226.c	1.0	1.0	1.0	1.0	[0]	3.0
student4959.cstudent1120.c	0.23076923076923078	0.6068376068376068	0.6068376068376068	0.2857142857142857	[0	2.0
student4959.cstudent1326.c	0.23076923076923078	0.6068376068376068	0.6068376068376068	0.2857142857142857	[0	2.0
student4959.cstudent5226.c	0.23076923076923078	0.6068376068376068	0.6068376068376068	0.2857142857142857	[0	2.0
student5226.cstudent1120.c	1.0	1.0	1.0	1.0	[0]	3.0
student5226.cstudent1326.c	1.0	1.0	1.0	1.0	[0]	3.0
student5226.cstudent4959.c	0.23076923076923078	0.6068376068376068	0.6068376068376068	0.2857142857142857	[0	2.0
student1013.cppstudent1016.cpp	0.8349056603773585	0.8158991565644992	0.8895394939386995	0.8591885441527446	, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0,	32.0
student1013.cppstudent1029.cpp	0.014354066985645933	0.5587453482190324	0.5587453482190324	0.01904761904761905	[0	2.0
student1013.cppstudent1044.cpp	0.05741626794258373	0.6564327485380117	0.6564327485380117	0.1004566210045662	[0, 1, 0,	10.0
student1016.cppstudent1013.cpp	0.8349056603773585	0.8158991565644992	0.8895394939386995	0.8591885441527446	, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0,	32.0
student1016.cppstudent1029.cpp	0.014150943396226415	0.5587002096436059	0.5587002096436059	0.018779342723004695	[0	2.0
student1016.cppstudent1044.cpp	0.05660377358490566	0.5804269106155898	0.5804269106155898	0.0990990990990991	0, 1, 0, 0	10.0
student1029.cppstudent1013.cpp	0.014354066985645933	0.5587453482190324	0.5587453482190324	0.01904761904761905	[0	2.0
student1029.cppstudent1016.cpp	0.014150943396226415	0.5587002096436059	0.5587002096436059	0.018779342723004695	[0	2.0

Figure 20: Screenshot of 'Numerical score' table with Similarity Techniques



filenames	otherops_lev	otherops_jaro	otherops_jarow	otherops_dice	otherops_lcb	otherops_lcblength
student1120.cstudent1326.c	0.7777777777777778	0.8842592592592592	0.9305555555555555	0.8736842105263158	,), {, (,), }, {, (,), (,), (,), }, }]	43.0
student1120.cstudent4959.c	0.8222222222222222	0.8590964590964592	0.9154578754578755	0.8255813953488372	[(,), {, (,), (,), (,),	28.0
student1120.cstudent5226.c	1.0	1.0	1.0	1.0	[(,), {, (,), (,), (,), {, (,), }, (,), {, (...	84.0
student1326.cstudent1120.c	0.7777777777777778	0.8842592592592592	0.9305555555555555	0.8736842105263158	,), {, (,), }, {, (,), (,), (,), }, }]	43.0
student1326.cstudent4959.c	0.7870370370370371	0.8481481481481482	0.908888888888889	0.9081632653061225	, (,), (,), (, (,), (,),),	32.0
student1326.cstudent5226.c	0.7777777777777778	0.8842592592592592	0.9305555555555555	0.8736842105263158	,), {, (,), }, {, (,), (,), (,), }, }]	43.0
student4959.cstudent1120.c	0.8222222222222222	0.8590964590964592	0.9154578754578755	0.8255813953488372	[(,), {, (,), (,), (,),	28.0
student4959.cstudent1326.c	0.7870370370370371	0.8481481481481482	0.908888888888889	0.9081632653061225	, (,), (,), (, (,), (,),),	32.0
student4959.cstudent5226.c	0.8222222222222222	0.8590964590964592	0.9154578754578755	0.8255813953488372	[(,), {, (,), (,), (,),	28.0
student5226.cstudent1120.c	1.0	1.0	1.0	1.0	[(,), {, (,), (,), (,), {, (,), }, (,), {, (...	84.0
student5226.cstudent1326.c	0.7777777777777778	0.8842592592592592	0.9305555555555555	0.8736842105263158	,), {, (,), }, {, (,), (,), (,), }, }]	43.0
student5226.cstudent4959.c	0.8222222222222222	0.8590964590964592	0.9154578754578755	0.8255813953488372	[(,), {, (,), (,), (,),	28.0

Figure 21: Screenshot of 'Other Operators' score table with Similarity Techniques

Given all the figures of single tables of token categories, it is observed that all similarity algorithms have been executed on a set of keywords, set of identifiers and all other sets. Further computation involves the aggregation of current values to derive an average score for a single comparison of two files.

3.4.2 Data Aggregation and Results

Referring to the last subsection about similarity algorithm execution, all the individual score columns for each token category is aggregated to achieve a final aggregated score for a token category. For example, all the single keyword score table columns can be aggregated to form one 'keyword' score for the two files. All other token categories follow the same aggregation procedure. A simple representation of aggregating similarity measures for one token category is shown below as follows:

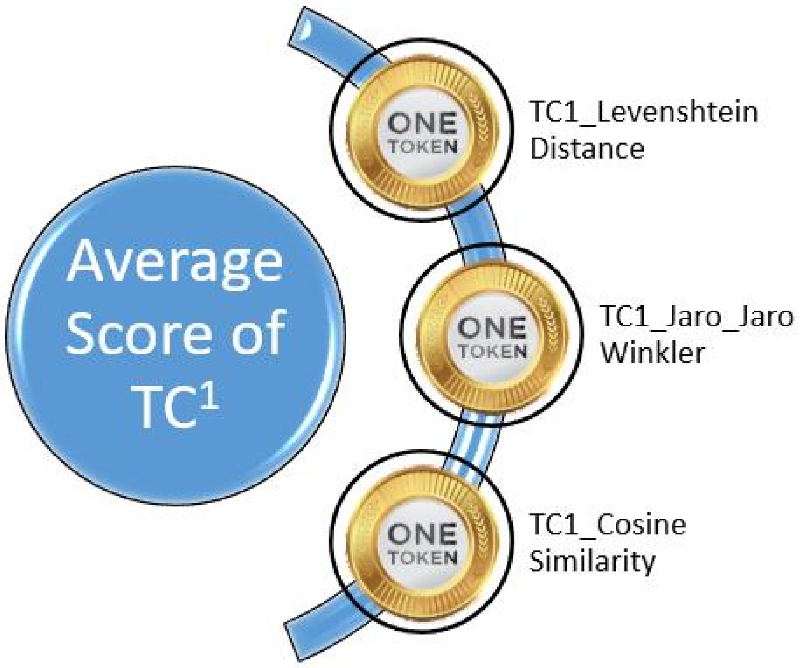
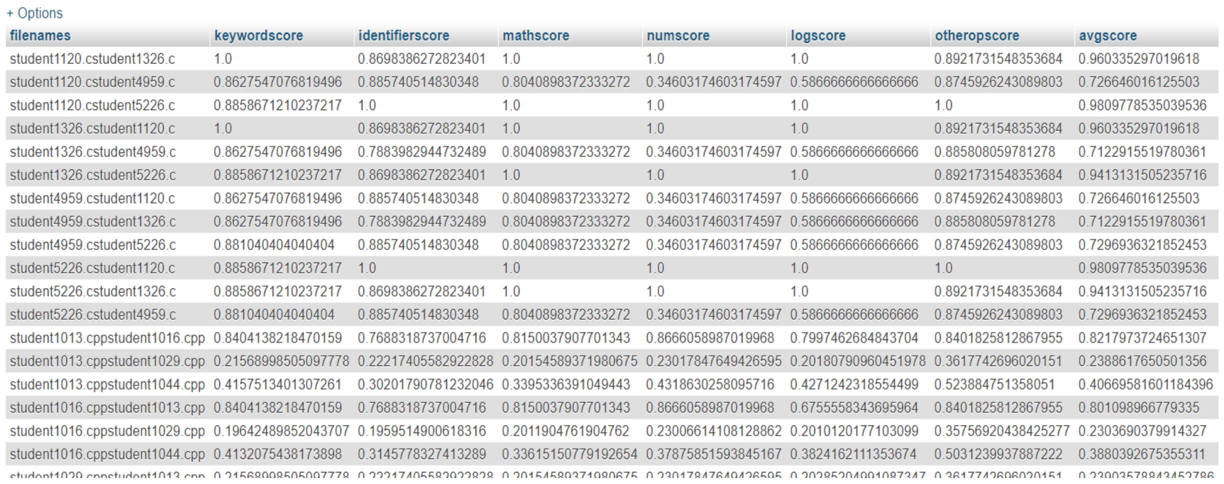


Figure 22: Aggregation of similarity measures for a token category (TC)

The aggregated scores for all the token categories include keywords, identifiers, numerical values, math operators, logical operators, and other operators. In the figure given above, TC stands for token category and can be any of the abovementioned. The snapshot of the average scores for the files is shown below as follows:


+ Options

filenames	keywordscore	identifierscore	mathscore	numscore	logscore	otheropscore	avgscore
student1120.cstudent1326.c	1.0	0.8698386272823401	1.0	1.0	1.0	0.8921731548353684	0.960335297019618
student1120.cstudent4959.c	0.8627547076819496	0.885740514830348	0.8040898372333272	0.34603174603174597	0.5866666666666666	0.8745926243089803	0.726646016125503
student1120.cstudent5226.c	0.8858671210237217	1.0	1.0	1.0	1.0	1.0	0.9809778535039536
student1326.cstudent1120.c	1.0	0.8698386272823401	1.0	1.0	1.0	0.8921731548353684	0.960335297019618
student1326.cstudent4959.c	0.8627547076819496	0.7883982944732489	0.8040898372333272	0.34603174603174597	0.5866666666666666	0.885808059781278	0.7122915519780361
student1326.cstudent5226.c	0.8858671210237217	0.8698386272823401	1.0	1.0	1.0	0.8921731548353684	0.9413131505235716
student4959.cstudent1120.c	0.8627547076819496	0.885740514830348	0.8040898372333272	0.34603174603174597	0.5866666666666666	0.8745926243089803	0.726646016125503
student4959.cstudent1326.c	0.8627547076819496	0.7883982944732489	0.8040898372333272	0.34603174603174597	0.5866666666666666	0.885808059781278	0.7122915519780361
student4959.cstudent5226.c	0.881040404040404	0.885740514830348	0.8040898372333272	0.34603174603174597	0.5866666666666666	0.8745926243089803	0.7296936321852453
student5226.cstudent1120.c	0.8858671210237217	1.0	1.0	1.0	1.0	1.0	0.9809778535039536
student5226.cstudent1326.c	0.8858671210237217	0.8698386272823401	1.0	1.0	1.0	0.8921731548353684	0.9413131505235716
student5226.cstudent4959.c	0.881040404040404	0.885740514830348	0.8040898372333272	0.34603174603174597	0.5866666666666666	0.8745926243089803	0.7296936321852453
student1013.cppstudent1016.cpp	0.8404138218470159	0.7688318737004716	0.8150037907701343	0.8666058987019968	0.7997462684843704	0.8401825812867955	0.8217973724651307
student1013.cppstudent1029.cpp	0.21568998505097778	0.22217405582922828	0.20154589371980675	0.23017847649426595	0.20180790960451978	0.3617742696020151	0.2388617650501356
student1013.cppstudent1044.cpp	0.4157513401307261	0.30201790781232046	0.3395336391049443	0.4318630258095716	0.4271242318554499	0.523884751358051	0.40669581601184396
student1016.cppstudent1013.cpp	0.8404138218470159	0.7688318737004716	0.8150037907701343	0.8666058987019968	0.6755558343695964	0.8401825812867955	0.801098966779335
student1016.cppstudent1029.cpp	0.19642489852043707	0.1959514900618316	0.2011904761904762	0.23006614108128862	0.2010120177103099	0.35756920438425277	0.2303690379914327
student1016.cppstudent1044.cpp	0.4132075438173898	0.3145778327413289	0.33615150779192654	0.37875851593845167	0.3824162111353674	0.5031239937887222	0.3880392675355311
student1029.cppstudent1013.cpp	0.21568998505097778	0.22217405582922828	0.20154589371980675	0.23017847649426595	0.20285204991087347	0.3617742696020151	0.23903578843452786

Figure 23: Screenshot of ‘Average scores’ table for student assignments

The classification or the process of determining whether the two assignments are plagiarized or not can be verified by the average score. The results decided were in the favor of plagiarism and the range for classification is given in the table below:

Table 10: Classification rules for Plagiarism detection

Average score	Class (Plagiarism level)
If average score < 0.70	Low Plagiarism
If average score between 0.70 AND 0.85	Average Plagiarism
If average score > 0.85	High Plagiarism
If average score == 1	Full Plagiarism
If average score == 0	No Plagiarism

The last two classification rules, which say that two assignments are said to be ‘exactly similar’ or ‘fully plagiarized’ if the average score is ‘1’ and the last rule where average score is found out to be ‘0’, indicates that two assignments are unique and have no similar content at all.

It would be inappropriate to conclude that assignments attaining an average score of ‘0.40’ share similar content / plagiarized. This decision usually depends on the assignment evaluator or the professor of the subject to declare a 10% similarity as a ‘cheating case’. In many cases use of one word multiple times could result in 10-15% similarity with an average score of around 0.40 to 0.50 and could lead to a cheating case even though the assignments have not been copied. There is an equal level of certainty between the boundaries of ‘unique’ and ‘smartly plagiarized’ assignments. There are cases where a student can smartly alter his/her assignment according to another student’s assignment by following obfuscation techniques such as changing program blocks (up, down), spamming spaces between lines, adding single and multiple line comments, moving indentations in the code and other non-novice programmer approaches. It is highly advised that the evaluator should personally view the codes for the Low and Average plagiarism level assignments and make a decision accordingly. The proposed ‘SimDec’ similarity detector forensic engine overpowers the novice programmer techniques such as:

- Flooding whitespaces in the code for a quick escape from plagiarism
- Block comments
- Renaming variables
- Shifting the Indentation of the code
- Changing positions of program blocks
- Other minor approaches

3.5 Third Phase – Representation

This last phase of the system development in which the information or data acquired is displayed to the end-users in the form of analysis and visualizations. The visuals of information such as average score, similarity measures result, and other computations represented on the front screen aids the user’s decision-making process. The software system’s architecture is primarily based on a three-tier model such as database, model-driven code and front-end GUI. More details about the development is defined in the next section.

3.5.1 Full-stack Development

Full-stack system development is a development process in which there is an involvement of client and server as well. Through a user interface component, the client queries or orders a fetch operation to and from the database, and server processes the request. The basic understanding of programming languages required for full-stack development are HTML, CSS and PHP. There are other client and server programming languages such as ASP, Node.js, Angular.js and others, but the knowledge of basic concepts helps to understand the advanced languages.

WAMP / XAMP - The XAMP / WAMP stack stands for Windows-Apache Server – MySQL – PHP. This is the best suitable stack for our proposed system in this research because of its benefits such as compatibility with Java Programming Language (JPL) and environment integration. The web application has been developed using HTML, CSS, JSP, PHP, and other client & server side languages like JQuery and AJAX. For the local instance, the systems front-end is connected to the IDE through server side PHP and MySQL intermediate layers. The input is given in the code and the computed values are stored in the MySQL database (JDBC connection/ plugin). The front-end queries and fetches values from the same database and that's how the full-stack development is successfully achieved.

Intellij IDEA – The integrated development environment (IDE) used for building the similarity detector engine was Intellij IDEA and not Eclipse because of a few advantages of Intellij such as productive Java coding experience, smart coding features, smooth integration of ANTLR plugin, convenient connection establishment with the MySQL database and millions of built-in tools and supported frameworks.

Plugins – The external plugins imported in the java project were ANTLR and JDBC connectors for tokenization and connectivity with MySQL purposes. Other apache plugins such as common text was kept for alternative approaches. ANTLR Plugin available at https://plugins.jetbrains.com/plugin/7358-antlr-v4-grammar-plugin. JDBC plugin available at https://dev.mysql.com/downloads/connector/j/

MySQL Database – The relational MySQL database, which is also a part of XAMPP package is chosen as the database for storing and loading values. The data extracted from source codes is stored in the Db tables and accessed by the user at the front-end.

3.5.2 Visualization on the Web GUI

The visual infographics of the data acquired from the comparison is stored and displayed on the front-end via a web application. Figure 24 represents the main homepage of the SimDec web application. The figure following the first one below is navigated through 'Analysis' page from the homepage's menu bar. The analysis page shows the paginated views for displaying low, average and high student assignments score.

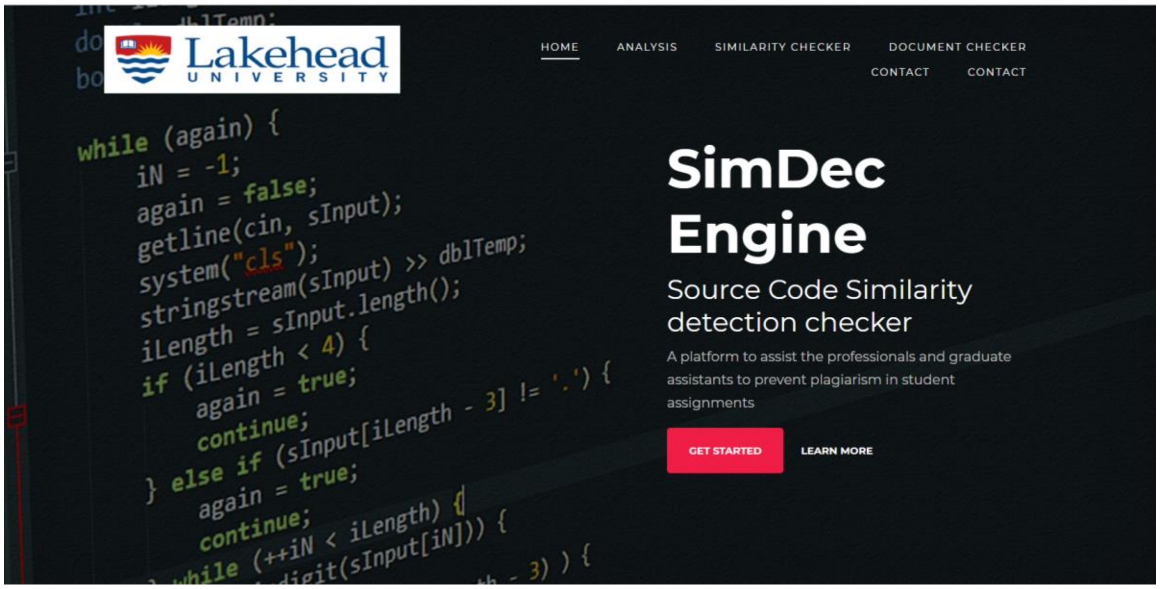
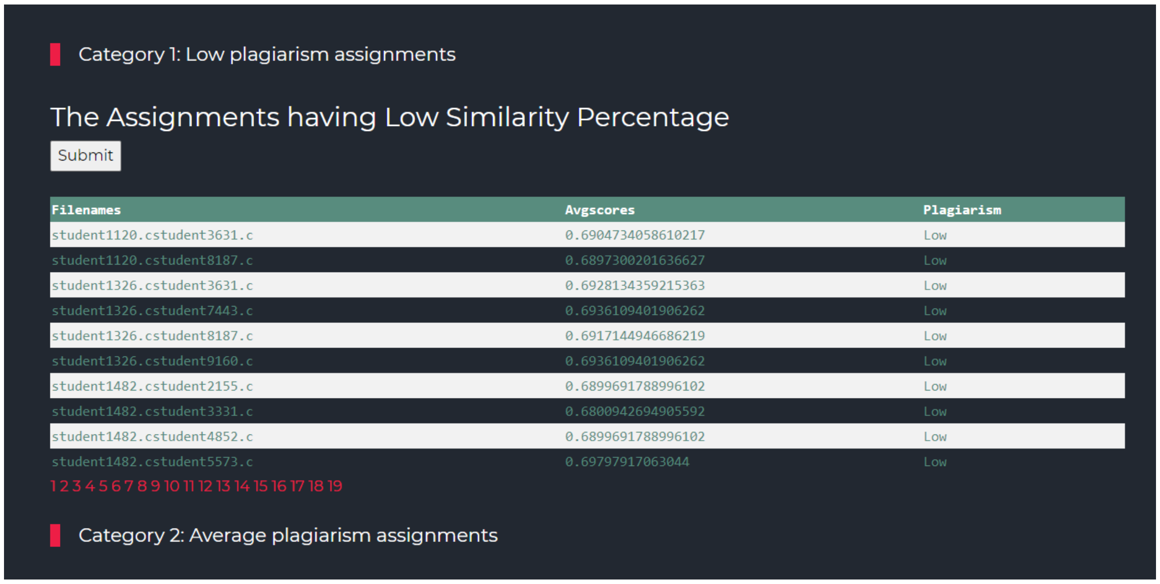

Lakehead UNIVERSITY
HOME
ANALYSIS
SIMILARITY CHECKER
DOCUMENT CHECKER
CONTACT
CONTACT
SimDec Engine
Source Code Similarity detection checker
A platform to assist the professionals and graduate assistants to prevent plagiarism in student assignments
GET STARTED
LEARN MORE

Figure 24: SimDec Web application - HomePage



Category 1: Low plagiarism assignments

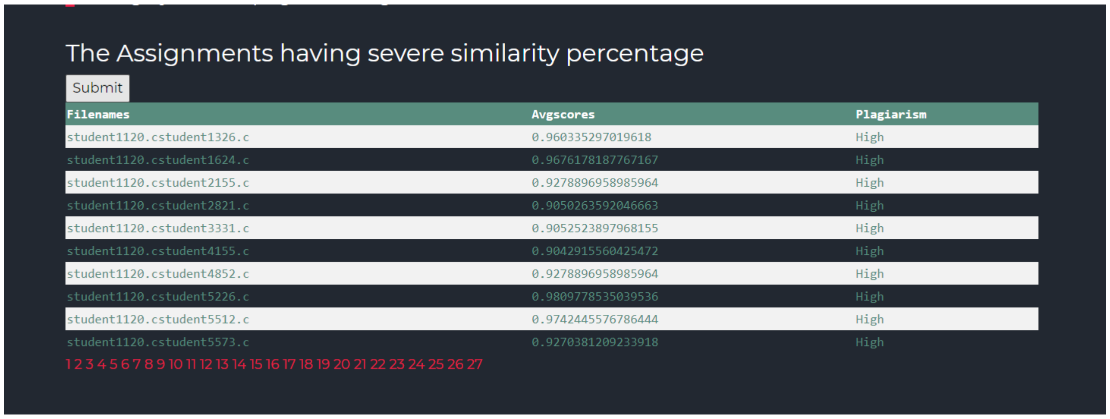
The Assignments having Low Similarity Percentage

Filenames	Avgscores	Plagiarism
student1120.cstudent3631.c	0.6904734058610217	Low
student1120.cstudent8187.c	0.6897300201636627	Low
student1326.cstudent3631.c	0.6928134359215363	Low
student1326.cstudent7443.c	0.6936109401906262	Low
student1326.cstudent8187.c	0.6917144946686219	Low
student1326.cstudent9160.c	0.6936109401906262	Low
student1482.cstudent2155.c	0.6899691788996102	Low
student1482.cstudent3331.c	0.6800942694905592	Low
student1482.cstudent4852.c	0.6899691788996102	Low
student1482.cstudent5573.c	0.69797917063044	Low

Category 2: Average plagiarism assignments

Figure 25: Assignment Level- Low, Average, High Display

The figure given below displays the assignments having ‘Severe’ or ‘Critical’ plagiarism score. That is, average score above 0.85 to 1.

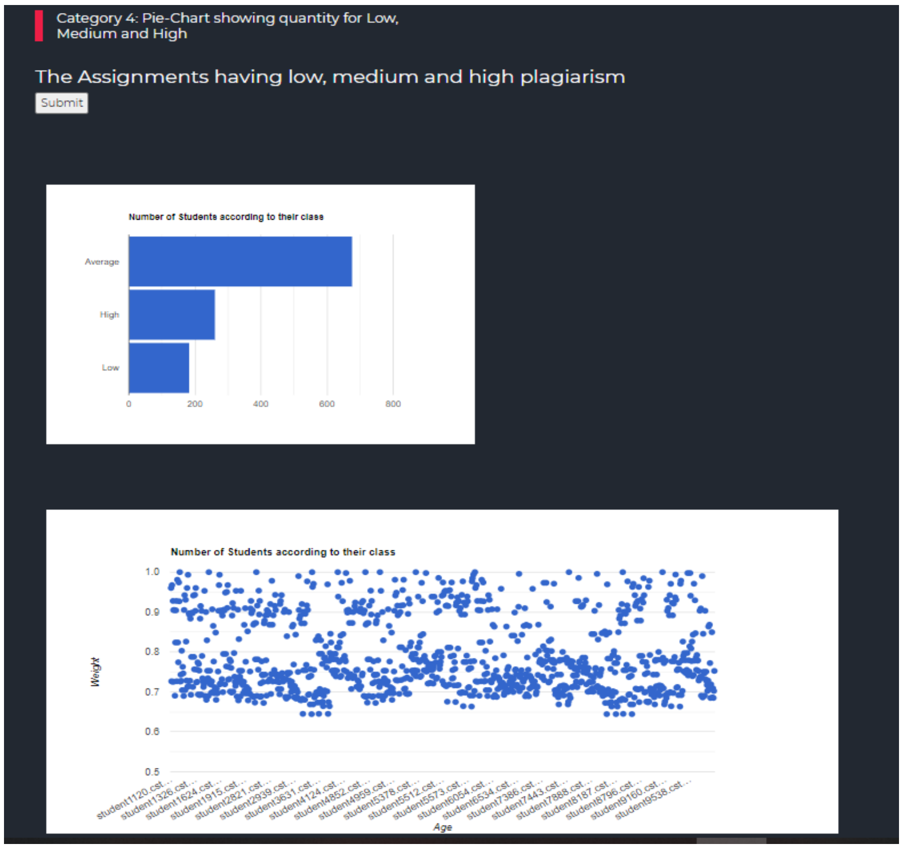


The Assignments having severe similarity percentage

Filenames	Avgscores	Plagiarism
student1120.cstudent1326.c	0.960335297019618	High
student1120.cstudent1624.c	0.9676178187767167	High
student1120.cstudent2155.c	0.9278896958985964	High
student1120.cstudent2821.c	0.9050263592046663	High
student1120.cstudent3331.c	0.9052523897968155	High
student1120.cstudent4155.c	0.9042915560425472	High
student1120.cstudent4852.c	0.9278896958985964	High
student1120.cstudent5226.c	0.9809778535039536	High
student1120.cstudent5512.c	0.9742445576786444	High
student1120.cstudent5573.c	0.9270381209233918	High

Figure 26: SimDec Engine - Severe Similarity Display

The end of the ‘Analysis’ page shows the bar-graph, pie-chart and scatterplot representing ‘Low’, ‘Average’ and ‘High’ percentages (quantity) student assignments.

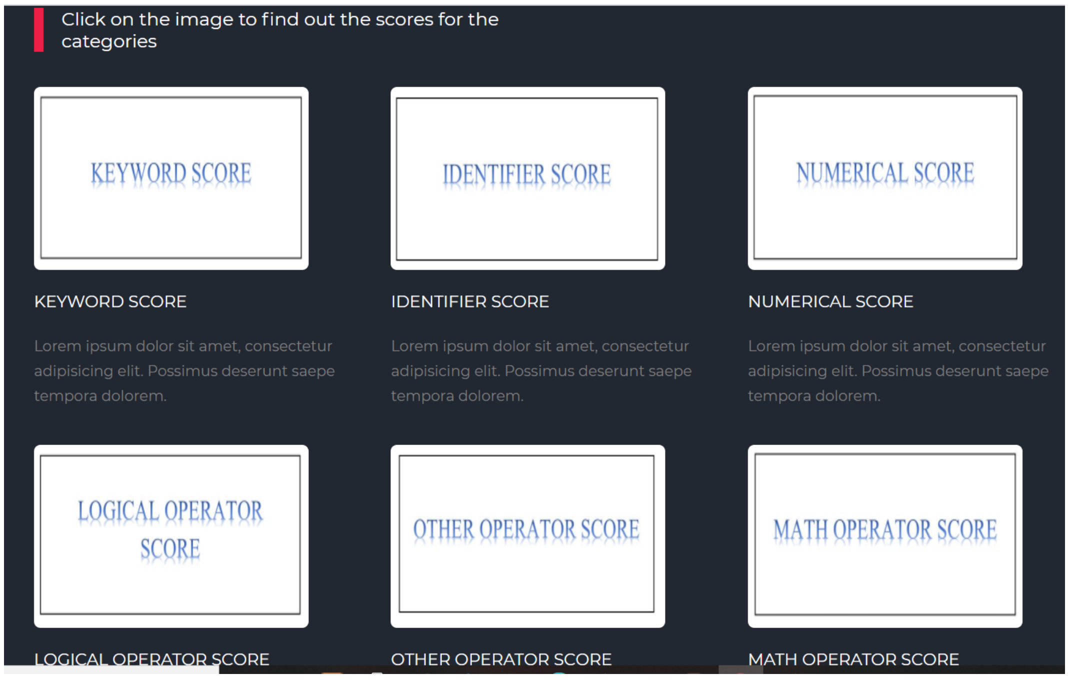

Category 4: Pie-Chart showing quantity for Low, Medium and High

The Assignments having low, medium and high plagiarism

Submit

Number of Students according to their class
Average
High
Low
0
200
400
600
800
Number of Students according to their class
Weight
Age

Figure 27: SimDec Web App Visualization - Bar Graph & Charts


Click on the image to find out the scores for the categories

KEYWORD SCORE

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Possimus deserunt saepe tempora dolorem.

IDENTIFIER SCORE

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Possimus deserunt saepe tempora dolorem.

NUMERICAL SCORE

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Possimus deserunt saepe tempora dolorem.

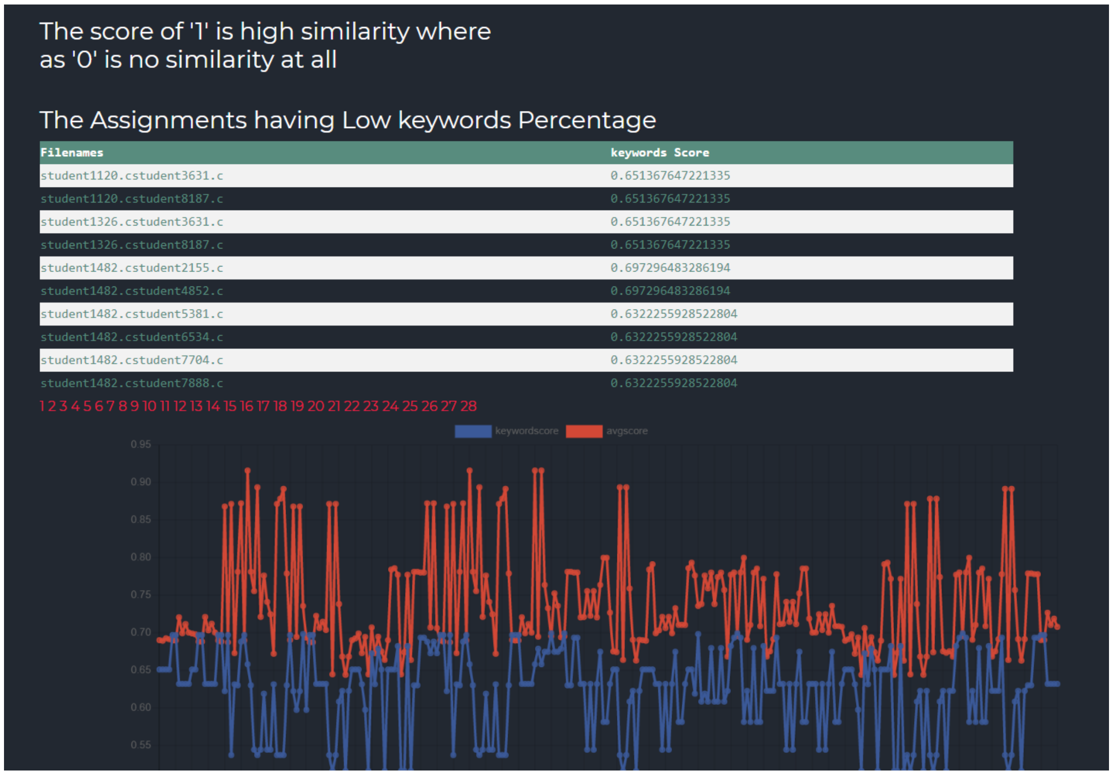
LOGICAL OPERATOR SCORE

OTHER OPERATOR SCORE

MATH OPERATOR SCORE

Figure 28: SimDec Token Categories

The ‘Similarity Checker’ page, accessible from all the website pages, shows a set of div tags for each of the token categories such as keyword score, identifier score, numerical score, logical operator score, other, math operator score. Every token category has a page within displaying low, average and high for respective token category score. The token category analysis is shown by visual representations using visualization tools such as Google charts, ChartJs, trial version of FusionCharts. A new advanced feature added to the search facility is the dropdown listing all the student file records. Upon selecting, the similarity measures such as Levenshtein distance, Jaro, Jaro-W inkler and dice coefficient for all token categories on the respective pages. Figure 30 shows the dropdown feature and display of similarity measures upon accessing the submit button for any student assignment pair.

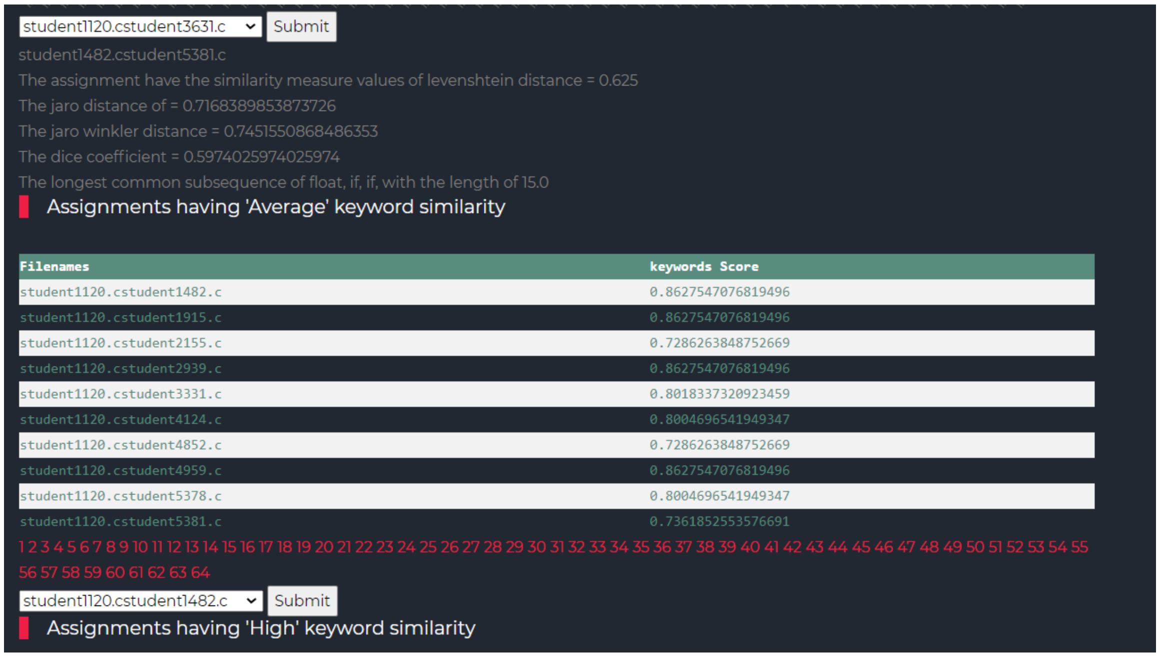

The score of '1' is high similarity where
as '0' is no similarity at all

The Assignments having Low keywords Percentage

Filenames	keywords Score
student1120.cstudent3631.c	0.651367647221335
student1120.cstudent8187.c	0.651367647221335
student1326.cstudent3631.c	0.651367647221335
student1326.cstudent8187.c	0.651367647221335
student1482.cstudent2155.c	0.697296483286194
student1482.cstudent4852.c	0.697296483286194
student1482.cstudent5381.c	0.6322255928522804
student1482.cstudent6534.c	0.6322255928522804
student1482.cstudent7704.c	0.6322255928522804
student1482.cstudent7888.c	0.6322255928522804

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28

Figure 29: Token Category Percentage with Visualization



student1482.cstudent5381.c
The assignment have the similarity measure values of levenshtein distance = 0.625
The jaro distance of = 0.7168389853873726
The jaro winkler distance = 0.7451550868486353
The dice coefficient = 0.5974025974025974
The longest common subsequence of float, if, if, with the length of 15.0

Assignments having 'Average' keyword similarity

Filenames	keywords Score
student1120.cstudent1482.c	0.8627547076819496
student1120.cstudent1915.c	0.8627547076819496
student1120.cstudent2155.c	0.7286263848752669
student1120.cstudent2939.c	0.8627547076819496
student1120.cstudent3331.c	0.8018337320923459
student1120.cstudent4124.c	0.8004696541949347
student1120.cstudent4852.c	0.7286263848752669
student1120.cstudent4959.c	0.8627547076819496
student1120.cstudent5378.c	0.8004696541949347
student1120.cstudent5381.c	0.7361852553576691

Assignments having 'High' keyword similarity

Figure 30: Similarity Measures scores representation to users

We have discussed the systematic workflow of our 'SimDec' forensic engine for 'C' & 'C++' source code plagiarism detection in this chapter with the help of block diagrams and system architecture. The above given figures in section 3.5.2 of this chapter represents the web user interface prototype for our system. The dashboard intends to display the findings or analytics of the experimentation in form of visualization and eases the process of identifying source-code thieves for the user evaluating the assignments. The next chapter focuses on the mathematical algorithms or similarity measures embedded in our system to calculate the similarity percent in the source codes.

Chapter 4. Mathematical Similarity Algorithms

In this chapter, we are going to emphasize on the mathematical similarity detection algorithms or techniques that are favorable for finding similarities between two words or strings. A token in general, could be a word like 'hello', number like '9', or just a character like '<'. There are different categories of tokens as mentioned in section 3.1 and section 3.3 respectively. The types of similarity measures mentioned in this chapter are embedded in the 'SimDec' systems second phase called 'Tokenizer customization' where the similarity measures are executed on the sets of tokens of two files to find the percent of similarity. If taken a close observation at table 1 from chapter 2, the previous plagiarism detection (PD) and clone detection (CD) tools like YAP3, nicad and iclones made use of Karp Robin, token matching and cosine similarity measures for finding similarities in source codes. These previous tools used one or a combination of maximum two existing techniques to detect similarities and therefore, leading it to a weak detection strategy. In our proposed system, we have used multiple similarity measures to gather enough evidence to categorize a source-code comparison pair as plagiarized or not plagiarized. This approach of using multiple measures leads to heavy computing time but that is where supervised machine learning jumps in to predict the category and reduce time complexity. This chapter explains the selected similarity algorithms/measures used with our systems configuration such as levenshtein distance, Jaro and Jaro-Winkler distance measure, Sorensen dice coefficient and cosine similarity. All these techniques fall in different categories such as distance based, token based and sequence based methods described below as individual sections.

4.1 Edit Distance based Algorithms

According to the work-flow explained in the section ' First phase: Lexical Analysis' of the chapter 3 of this thesis, the source code breaks down into number of lexemes / tokens and is forwarded to the tokenizers deployed within the lexical analysis phase. The tokens are mostly strings, integers, characters, and operators stored in separate containers or cluster sequences after the tokenizers categorize them into keywords, math operators, numeric operators and others. The distinct sequences from containers in the file comparison stage are concatenated together in one sequence. They are evaluated with distance similarity algorithms to compute the similarity distance between the string sequences. In this research, we have used this particular levenshtein distance method to measure the difference between two strings or two sequences acquired from the tokenization process. The measure has proven to be effective on two sequences of for example, keywords extracted from two source codes and find the distance between them. This section describes the potential distance-similarity algorithm such as Levenshtein distance taken into consideration for this research because of its adaptability to work with strings of unequal lengths.

4.1.1 Levenshtein Distance

Levenshtein distance is a distance similarity method invented by mathematician Vladimir Levenshtein in the year 1965 [28]. Levenshtein distance, also called edit distance, is defined as the similarity between two string sequences 1' and's 2'. The algorithm focuses on the minimum number of changes required to convert string's 1' into string's 2' with an operation such as insertion and deletion in string's 1'. The algorithm can be illustrated in the programming area as lev (s1, s2) where the value lies between 0 and 1. The values closer to '0' indicate less similarity and nearer or equal to '1' indicate a greater measure of similarity. For example, lev (hello, hell) will fall somewhere between 0.8 and 1 as just one letter of's 1' is missing in's 2'. The mathematical equation for Levenshtein distance is given below as equation (1):

$$\underset{a,b}{lev}(i,j) = \int_{0}^{\max(i,j)} \min \int \begin{matrix} \underset{a,b}{\text{lev}}(i-1,j)+1 \\ \underset{a,b}{\text{lev}}(i,j-1)+1 \\ \underset{a,b}{\text{lev}}(i-1,j-1)+1_{(ai\neq bj)} \end{matrix}$$

In the above equation, 1(ai≠bi) is the indicator function set to zero initially and equal to 1 otherwise. Lev$_{(a,b)}$(i,j) = distance between first *i* characters of string ‘a’ and first *j* characters of string ‘b’. The best example for Levenshtein distance between 'HONDA' and 'HYUNDAI' is 3 and edit changes using insertion, substitution, and deletion operations. The wider applications of Levenshtein distance in string matchings falls under dynamic programming, and the pseudocode for DP approach for Levenshtein distance is given below:

```
int LevenshteinDistance(char s[1..m], char t[1..n])
  // d is a table with m+1 rows and n+1 columns
  declare int d[0..m, 0..n]

   for i from 0 to m
```

```
        d[i, 0] := i
    for j from 0 to n
        d[0, j] := j

    for i from 1 to m
        for j from 1 to n
        {
            if s[i] = t[j] then cost := 0
                           else cost := 1
            d[i, j] := minimum(
                                 d[i-1, j] + 1,     // deletion
                                 d[i, j-1] + 1,     // insertion
                                 d[i-1, j-1] + cost   // substitution
                             )
        }

    return d[m, n]
```

4.1.2 Jaro Distance

Jaro edit-distance method is a similarity measure invented by scientist Mathew A. Jaro [29]. Like any other algorithm, Jaro similarity measures the distance between two string sequences. The value of Jaro (s1, s2) mostly ranges between 0 to 1, where two strings are equal when the value is 1 and not equal at all when a value is zero. The mathematical formula for Jaro proposed by Mathew Jaro [28] and a detailed explanation on value calculation is given under as follows as equation 2:

$$Jaro\ Similarity = \begin{cases} 0, if\ x = 0 \\ \frac{1}{3}\left(\frac{x}{|s1|} + \frac{x}{|s2|} + \frac{x-t}{x}\right), & for\ x! = 0 \end{cases}$$

From the equation above,

x = number of matching characters,

t = half the number of transpositions,

|s1| and |s2| = lengths of string s1 and s2

The matches are accurate if they are not farther than $\left[\frac{\max(|s1|,|s2|)}{2}\right] - 1$ and t = half the number of characters in both strings in a different order.

Consider s1 = ‘rover’ and s2 = ‘flower’, here the matching characters is three such as [o, e, r] in a different order. Number of characters not in order = 4 i.e. In s2 = [f, l, o, w]. Therefore, t = 4/2 = 2. From the above equation 2, Jaro similarity can be calculated as = 1/3 * ((3/5 + 3/6 + (3-2)/3) = 0.4719. The strings ‘rover’ and ‘flower’ have a Jaro similarity measure of 0 < 0.4719 < 1. The purpose of using Jaro and Jaro-Winkler edit-distance measure is to support the results obtained from Levenshtein method. Suppose for two sequences, levenshtein distance result obtained is ‘0.70’ and Jaro and Jaro-Winkler distance measure give out the score which ranges between 0.60 – 0.75; then we have multiple measures giving out the similar result for two sequences.

4.1.3 Jaro- Winkler Distance

Following the invention of Jaro distance measure, William Winkler [30] proposed an upgrade to the Jaro metric called as ‘Jaro-Winkler’. The Jaro-Winkler distance measure is similar to the Jaro algorithm in most cases; the prefix of both the strings doesn’t match. They both produce different values when the prefix of both the strings don't match. The prefix scale 'p' in Winkler gives more accurate answers when strings have a common prefix of length 'L’. The Jaro-Winkler similarity measure is defined as follows in equation 3:

JW = J + Sf * L*(1 - J)

Where, J = Jaro measure obtained from above block,

Sf = scaling factor (0.1 by default),

L = length of matching prefix (max 4 characters long). Referring to the same string examples used for Jaro measure in the subsection above this one. Here for ‘rover’ and ‘lower’ we have L = 0.

The computation, JW = 0.4719 + 0.1 * 0 * (1 – 0.4719) = 0.4719. The Jaro-Winkler and Jaro in this case are equal. The results may be different for strings such as ‘Logitech’ and ‘Lotto’, where L =2.

4.2 Token-based Algorithms

4.2.2 Sorenson Dice Coefficient

Thorvald Sorensen [31] and Lee Dice [32] independently developed the ‘Sorensen dice coefficient’ or ‘dice index’ statistical tool used to gauge the two samples' similarity. This invention intended to differentiate the similarity between two distinct sequences. Assume ‘1' and ‘2' to be two distinct data sequences and |s1| and |s2| be the same sets' cardinalities. The dice index /coefficient equals twice the number of elements common to both the sets divided by the sum of cardinality sets. The mathematical equation for DC or DI (Dice Index) is given below in equation 4 as follows:

$$DCS = \frac{2\ |s1\ \cap s2|}{|s1| + |s2|}$$

The only difference between the Jaccard coefficient and DCS is that Jaccard counts the true positives once in both denominator and numerator. DCS falls in between 0 and 1 for two discrete sets. The DCS for string similarities is a variance of the normal DCS form and uses bigrams of the strings for computation, as shown in equation 5:

$$DCS\ for\ Strings = \frac{2nb}{nx + ny}$$

Here, 'nb' is the number of bigrams found in both the strings and 'nx' & 'ny' denote the number of bigrams found in string X and Y, respectively. Consider the words' Deer' and 'Dear', the set of bigrams in each word world be as follows:

X = {de, ee, er}, Y = {de, er, ar}

The common bigram between both the strings is {de}. Therefore, the DCS we obtain after calculation by putting in equation (5) is (2.1) / (3 + 3) = 0.33. The score ‘0.33’ is near to ‘0’ and less than 0.50 therefore, the sequences ‘X’ and ‘Y’ have only one common element and the severity is not so high, given 0.33 as the dice coefficient gauge score.

4.2.3 Cosine Similarity-Based Method

Cosine similarity can be defined as a document similarity metric that is used to measure the similarities between two documents irrespective of the size. It measures the cosine of the angle between two vectors in a 2D multi-dimensional space. The vectors selected for measurement can be strings, arrays and value objects in a coded algorithm. The core programming language used for developing the proposed system is Java, and hence, forming the vectors from the tokenization approach is not cumbersome in the procedure. The main advantage of this method is it can conclude that two documents can be oriented together even if they're far apart because of size irregularities. Like the other techniques stated above, the result value of cosine similarity ranges between 0 and 1. The similarity percentage is less if the cosine angle is big and high when the angle is small. The cosine similarity is implemented for document similarity in two ways as described below:

Approach 1: Consider 'A' and 'B' as two document vectors and measure the cosine similarity angle between the two vectors to justify the similarity between two documents in the range of 0 to 1. This approach is favorable for the research, focusing on occurrences of a word for checking document similarity.

Approach 2: Tokenize the document to form categories for simplification and then concatenate the distinct features into one complete vector. Follow this procedure for all the documents and then calculate the cosine angle between the vectors. The result for this approach would be more effective than approach one as the vector would contain all distinct elements from all the categories. The mathematical formula for cosine similarity is given under as equation 6:

$$Cosine\ Similarity = \cos(\theta) = \frac{X.Y}{||X||||Y||} = \frac{\sum_{i=1}^{n} X_i Y_i}{\sqrt{\sum_{i=1}^{n} X_i^2}}.\ \frac{1}{\sqrt{\sum_{i=1}^{n} Y_i^2}}$$

In the equation above, 'X' and 'Y' are the two vectors of attributes and cosine similarity is represented as a dot product and magnitude. The result obtained from this formula will be '1' if the documents are clones and '0' if they're the opposite. In the case of IR (Information retrieval), the angle between two '*term*' vectors cannot be > 90 degrees. Gunawan et al. [32] in their research on finding text relevance via cosine similarity mentioned the use of cosine similarity measures to find the relevancy of a suitable topic in multiple documents. The authors divided the system implementation into three stages such as pre-processing (removing punctuations from documents, converting all text to lower-case, etc), intermediate (keyword weighing between 0 and 1) and the last stage involves cosine angle measure to give out relevancy in terms of '0' or '1'.

4.3 Sequence Based Algorithms

4.3.1 Least Common Subsequence

Another Dynamic Programming (DP) approach considered in this proposed similarity detection engine after Levenshtein distance is the longest common substring (LCS) for assuring string similarity without any resulting numerical value. David Maier mentioned the complexity of some problems on subsequences and super sequences in his research [34]. Being a DP implementation, this algorithm has a time complexity of O(nm) where space is utilized more than time. The definition of LCS is simple as it identifies a substring in is '1' and checks for the same in's 2'. The algorithm also has a functionality of keeping track of the substring's maximum length and displaying it on the console. An example for LCS detection is given below in words, as there is no exclusive statistical explanation for it in algorithms. k-common substring problem ϵ LCS (X, Y, m, n) = Max(LCSuff(X,Y,I,j)) where 1 <= i ,j <= m, n. Max(LCSuff) is the equation where both the strings lengths is reduced by 1 if the last characters match.

4.4 Integration with the System

The selected similarity measures described with details in the above sections have been implemented in java programming language with our 'SimDec' system. The single MySQL table images shown in figures 16 to 23 in the chapter 3 represent the similarity scores of each token category sequences such as keywords, identifiers, arithmetic operators, logical operators and other operators. The solo scores of each of the techniques such as Levenshtein distance, Jaro, Jaro-Winkler, Cosine similarity and Dice coefficient are aggregated and displayed as one average score of all token categories as shown in figure number 23. All these observations are recorded in the relational MySQL database at the back-end and this data is forwarded to the machine learning module of this research. SimDec system allocates one plagiarism category to all the files such as Low, Average and High. Low category could mean no plagiarism at all and should not be considered for personal evaluation of the source code by evaluators. Records with labels medium and High could mean that the plagiarism done is more than 50% of similar content and corresponding action shall be taken by the professors or evaluators. Concluding this chapter and moving on to the next chapter, we will discuss the computational performance analysis of all the supervised and unsupervised machine learning algorithms in the next chapter. Chapter 5 will represent a comparative study of all the algorithms and choose the technique that has better compatibility with the system-generated data.

Chapter 5. Computational Performance Analysis

This chapter focuses on the core concept of machine learning and its categories such as supervised and unsupervised machine learning. The data recorded on applying the similarity measures discussed in chapter 4 is stored in the relational database and exported to the spreadsheet view for working efficiently on the ML and python environment. The data primarily consists of class labels to each of the records assigned by our proposed system. The class label helps the supervised models to train and validation is impacted in a positive way. To deliver clustering and unsupervised analysis, the class label can be dropped and the process can resume. We have shortlisted a few ML algorithms to go on with at the beginning such as multi-class SVM, logistic regression and a simple neural networks for supervised learning and k-means, PCA from the unsupervised category of ML. Each of these mentioned algorithms with their evaluated model scores is described in the sections given below.

5.1 Machine Learning Algorithms

Machine learning is the child of artificial intelligence that automatically learns and improves from the programming experience. Machine Learning is used to develop computer programs that can access the data fed to it and learn the patterns on every run. The technology can learn the patterns on one dataset and implement them for another dataset. In simple terms, the model developed in python programming language can be executed on a dataset, save the model and then loaded later for testing purposes. The primary aim of machine learning remains to allow the computer to automatically learn the patterns without human assistance. Machine learning algorithms are often categorized as supervised, unsupervised and semi-supervised learning. More details on each of these categories will be given in the later sections.

5.1.1 Second Module of Research

Implementing machine learning algorithms on the data gathered from mining information from source code comparison. Our ‘SimDec’ system assigns a label for each comparison pair such as low, average and high as explained in the chapters above. The main purpose of making the system do it is to make it compatible with the supervised learning algorithms for prediction. Because without the class label, the generated data will only be good for unsupervised learning and not suitable for the classification or prediction. This information is stored in relational database systems and fetched by the users at the front-end. The labels generated in the dataset is required for applying supervised machine learning algorithms for achieving classification. The second module of this thesis is to extract and infiltrate the output of the SimDec software system and give it as an input to the machine learning algorithms. The development environments and platforms for both the modules are different and the configuration is independent. Both of the modules are not directly interlinked with each other and separated by the dataset. Dataset is the middle layer between software system and machine learning platform. This module involves supervised and unsupervised learning and the decision is concluded by observing and analyzing the results.

5.1.2 Programming & Development Environment

The development environment for both of the modules of this research is different, including programming languages and stack required for successfully achieving the technology implementation.

The IDE and programming languages for the first module of the research are Intellij IDEA and Java programming language, whereas, the same for the machine learning module is Google Colab/ Kaggle and Python programming language. Java is not very helpful and productive when it comes to data analysis and visualizations. Python was developed to fulfil the data science criteria and fit in the data visualization circle.

5.1.3 Need of ML in Software Systems

Since the invention of LISP and FORTRAN, ML has played a major role in software systems. The tools for building low-level and high-level programming languages didn’t change their layout or appearance, but are essentially the same. Look at the fancy editors, they have the features such as color highlighting, predicting next word when typing the current, and different programming styles. A software system is a mixture of source codes that is formulated in a flow to get a series of output. Adding machine learning and pattern recognition to the software code can enhance the system in numerous ways by increasing the code's efficiency. As language changes and usage shifts, new elements are discovered and the neural network can be revisited and retrained on the new data. To discuss the scope of machine learning in our thesis research, it is mainly implemented to decrease the running operation's time complexity. The time taken by the program to build and run the program could be reduced gradually as the number of computations would be reducing as well. To dive into the details, the number of computations required to assign a plagiarism level label to one comparison takes substantial time, starting from mathematical calculation to storing and loading from the dataset. Machine learning could easily omit the computing time as the model(s) could be trained on huge data and then can be used to predict the class label if the accuracy is reasonably well. This chapter will justify the above mentioned hypothesis by covering the machine learning algorithms and relevant concepts. The dataset selected for this experimentation is gullible with supervised and unsupervised for predictive and descriptive analysis both. There would be a comparative study observed in the below given sections and the best suitable technique for the system would be considered as a final analytical decision in this software engineering process. The figure given below mentions the features machine learning provides to enhance the software system code:

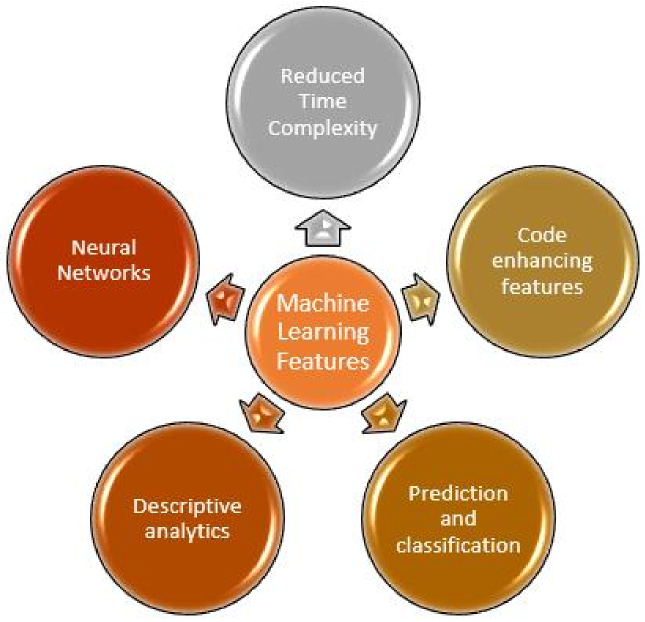


Figure 31: Machine Learning for Software systems

5.2 Supervised Learning Classification

The name ‘Supervised Learning’ is self-explanatory to mention the need and importance of this technique. Supervised techniques is like a teacher to guide a student and during the training process, the algorithms will search for patterns in the data correlate with the desired outputs. The nature of a supervised learning algorithm is to train on a set of data to get the patterns right and then predict the label or target variable for the newly presented data. Supervised learning can be split into two categories such as regression and classification. The regression approach is undertaken if a prediction is to be made for a continuous variable such as numerical scores, amount, percentage etc. Regression can be further sub-divided into linear and logistic where logistic regression is designed for categorical variable prediction and linear for numerical predictions. Classification algorithms are for predicting categorical labels such as high/low/medium, 0/1/2 or true/false in the datasets. The job is to simply take the input and assign a class or category that fits with the training data provided. A classification problem can be solved with a plethora of algorithms such as Support vector machines, Naïve Bayes, Decision trees, Neural Nets and K-Nearest Neighbor algorithms. We have considered three popular and quality algorithms for conducting our second research module such as SVM’s, Logistic regression and Neural Networks. The brief description of each one of the above mentioned is given in the lower sections.

5.2.1 Support Vector Machines (SVM)

Support Vector Machines (SVM) algorithm was developed by the authors [35][36] at the AT & T Bell laboratories in the year 1992. Support vector machines are supervised learning algorithms that can solve a classification problem using two-class (high, low) and multi-class (high, medium, and low). According to the data formulated in our system, a multi-class SVM was a favorable one as there are three class labels: high, average and low. The objective of a multi-class SVM is to find a hyperplane in an n-dimensional space that separates the data points according to their classes. The data points which are nearest to the hyperplane are called as ‘Support Vectors’. Another reason for SVM’s to be called kernelized vectors is because they convert input data space into a higher-dimensional space. The number of classifications required for one vs one multi-class classification can be found out by the formula given below:

$$\frac{n * (n - 1)}{2}$$

Kernel functions:

The popular kernel functions available in the scikit-learn are linear, polynomial, radial basis and sigmoid. The equations of the four functions are given below as:

Linear Function - $k(x_i, x_j) = x_i * x_j$

Polynomial function - $k(x_i, x_j) = (1 + x_i + x_j)d$

Radial Basis function (RBF Kernel) - $k(x_i, x_j) = exp(-\gamma||x_i - x_j||)^2$

Sigmoid function - $k(x_i, x_j) = \tanh(ax^T y + c)$

We will not discuss SVM's root contents by explaining the mathematical concept behind it as that knowledge is available on hundreds of websites online. The SVM algorithm performed efficiently on the SimDec data for multiclass classification and obtained an accuracy of 99% for both training and testing. The details of multi-class SVM results on our systems data are mentioned in short in the last section of this chapter.

5.2.2 Logistic Regression

Logistic regression is the simplest type of supervised Regressor used only when the target variable is categorical. Logistic regression was invented by a popular statician DR Cox as a binary probability model [37]. For numerical target variable, linear regression is recommended due to compatibility. An example would be to predict / classify whether the statement is true or false or yes or no. The function used in logistic regression is the sigmoid function and ranges from minus to plus variable as shown in the figure below:

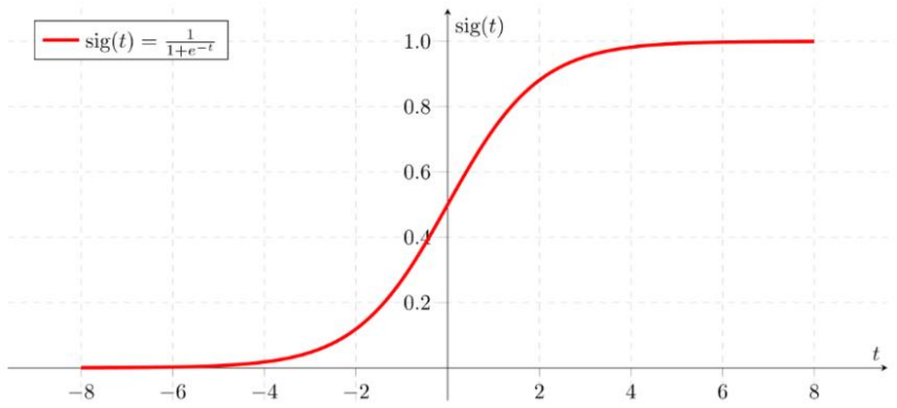


Figure 32: Sigmoid function for logistic regression

The analysis of hypothesis is the estimated probability used to infer how confident can predicted value be actual value when the given input is X. There are several types of logistic regressions such as binary logistic regression for predicting 0 or 1, multinomial logistic regression for classifying more than two labels such as high, average and low. A threshold needs to be set to predict the class of the data because the estimated probability is classified into classes. The cost function is to be considered in the main equation for linear regression and not logistic regression. MSE (Mean square error) is used with linear regression and if used with logistic, it will create a non-convex functions of parameters. The cost function for logistic regression is defined below in which, if y = 1, the output approaches to O as hΦ(x) approaches to 1. The cost to pay grows to infinity as hΦ(x) approaches to O. The same situation applies when y = 0, where there are bigger penalties when the label is y = 0 but algorithm predicts hΦ(x) = 1.

$$(Cost(h\emptyset(x), y) = \begin{cases} -\log\big(h\emptyset(x)\big) \ if\ y = 1 \\ -\log\big(1 - h\emptyset(x)\big) \ if\ y = 0 \end{cases}$$

The logistic regression algorithm implementation on the system data is described in section 5.4 of this chapter where the accuracy obtained with the same technique on system data is around 98% for training and testing the model.

5.2.3 Neural Networks

Warren McCullough and Pitts [38] at the University of Chicago invented neural networks as an activity that was derived from the calculus of ideas immanent in nervous activity. To give a short and simple description, a neural network is constructed with thousands of neurons and one neuron is a basic unit of the network. Neurons simply take the input, process the computation and give the output. For example, in a three-neuron neural network, the three inputs are multiplied with weights and added with a bias 'b' as shown below:

*x1 -> x1 * w1, x2-> x2*w2, x3->x3*w3* (where w1,w2,w3 are weights of the network)

Secondly, the inputs are added with a bias and passed through an activation function as follows:

$$Y = f(x1 * w1 + x2 * w2 + x3 + w3 + b)$$

The activation functions can be of 7 varying kinds such as sigmoid, ReLu, Tanh, linear activation, non-linear activation, softmax and swish. The detailed explanation about all the activation functions can be found out at https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/.

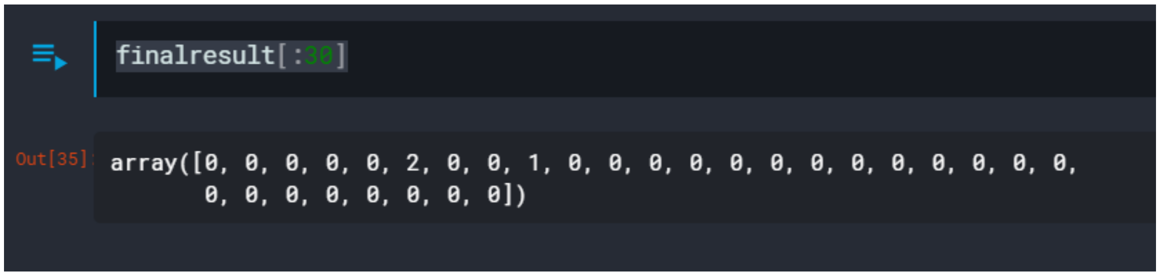
Getting back to neuron based neural networks, the setup can be established and coded in python using keras and tensorflow framework. The neural network could be built of several layers and activation functions to get the satisfactory result. The neural net model is supposed to be trained and tested on a dataset before putting it to actual real-world testing. The model is evaluated like all other machine learning models by standard loss and error computation. MSE (mean square error) is the loss function used for computing a neural network evaluation. Neural nets can be configured in three different ways such as feed-forward networks, backward propagation and ensemble learning (hybrid NN's). Lastly, we would like to discuss in brief about the optimizers, the optimizers available for balancing the weight and minimize the loss. The learning rate encompassed in the optimizers sleeps up the training time of the model. All these parameters can be altered and played with for getting an appropriate score. The code for model which we built for applying on our systems data is shown below as follows:

```
input_dim = len(data.columns) - 1
model = Sequential()
model.add(Dense(8, input_dim = input_dim , activation = 'relu'))
model.add(Dense(10, activation = 'relu'))
model.add(Dense(10, activation = 'relu'))
model.add(Dense(10, activation = 'relu'))
model.add(Dense(3, activation = 'softmax'))
model.compile(loss = 'categorical_crossentropy' , optimizer = 'adam' , metrics = ['accuracy'] )
model.fit(train_x, train_y, epochs = 10, batch_size = 20)
scores = model.evaluate(test_x, test_y)
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

There has been an imposition of four dense model layers consisting of three ReLu activation layers and one softmax layer. The loss function for compilation is categorical cross entropy with Adam optimizer for minimizing loss. The model is trained with training data (x,y) on 10 epochs and a batch size of 20 and then evaluated on testing data parameters (x,y) . The testing score achieved with 20 batch size is 93.22 % with a minimum loss of 0.1619. The model looks good at 93.22% accuracy on the current system data and therefore, a new data for another student assignment folder is given to the model for testing. The trained and tested model is saved, loaded back on the environment and tested on the data without the target variable. The one-hot encoding is done for the target variable ‘Plagiarism’ that is low, average and high as ‘0’, ‘1’, and ‘2’. The data is given to the model.predict(datafilename) and a numpy array is obtained as the predictions array. On printing the list, the predictions obtained for the data is shown below as follows:

```
finalresult=model.predict_classes(data1)
```

On printing - finalresult[:30]



```
finalresult[:30]
```

```
Out[35]: array([0, 0, 0, 0, 0, 2, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0, 0, 0, 0, 0])
```

Figure 33: Predictions for new system data: Neural Networks

5.3 Unsupervised Learning Clustering

Unsupervised machine learning is a type of machine learning technique in which there is no need to supervise the model. The data is fed to the model to discover the patterns and explore new information on the basis of the learning. Unsupervised learning mainly deals with data which has no label/target. Clustering analysis or descriptive mining holds the major portion of unsupervised ML. Other existing ML techniques are given as association rules and variation of association rules. There are various types of clustering algorithms such as Hierarchical clustering, K-means, KNN classification, principal component analysis, single value decomposition and independent component analysis. The study conducted on the SimDec system data involves two types of unsupervised clustering such as K-means and PCA due to the data's simple nature. The coding for both the techniques has been completed using python programming language in this module. The purpose of conducting unsupervised studies is to compare it with supervised learning and conclude the best ML category for the satisfying the second modules intention.

5.3.1 K-means Algorithm

James McQueen [39] proposed an algorithm for dataset instance into groups of clusters and the algorithm was named as ‘K-means clustering algorithm’. K-means clustering is an iterative clustering algorithm that helps you find the clustering's highest value by selecting a centroid / central points. The number of clusters ‘k’ are selected by coding the elbow method curve where the value of bent curve is selected as the desired number of clusters for the k-means operation. The entire data is clustered into ‘k’ groups and the output of the algorithm is a group of ‘labels’. The centroids are the hearts of the clusters and the bigger the cluster, the lower the granularity and the value of ‘k’. The small clusters often have large granularity and a bigger value of ‘k’. The techniques have been implemented in two languages such as Python and 'R'. Implementing the algorithms in 'R' programming language is because of aesthetic and efficient visualizations and easy data analytics. We are going to discuss a little bit of both the parts in this chapter. Dilpreet Singh and Chanda Reddy [40] gave a simple pseudocode for k-means algorithm in their paper and reciting the same as shown below:

The K-means Clustering Algorithm

1. Input data points 'D' and specify number of clusters 'K'
2. Initialize central points or centroids randomly
3. Associate each data point in 'D' with the nearest centroid. This will divide data points into 'k' clusters.
4. Recalculate the position of centroids and repeat the above two steps till final step
5. Represent data points with clusters

K-means with Python

The program starts with declaring imports and reading the data file as 'data' variable and then dropping the categorical attribute from the pandas frame, followed by the assigning the same categories to a label.

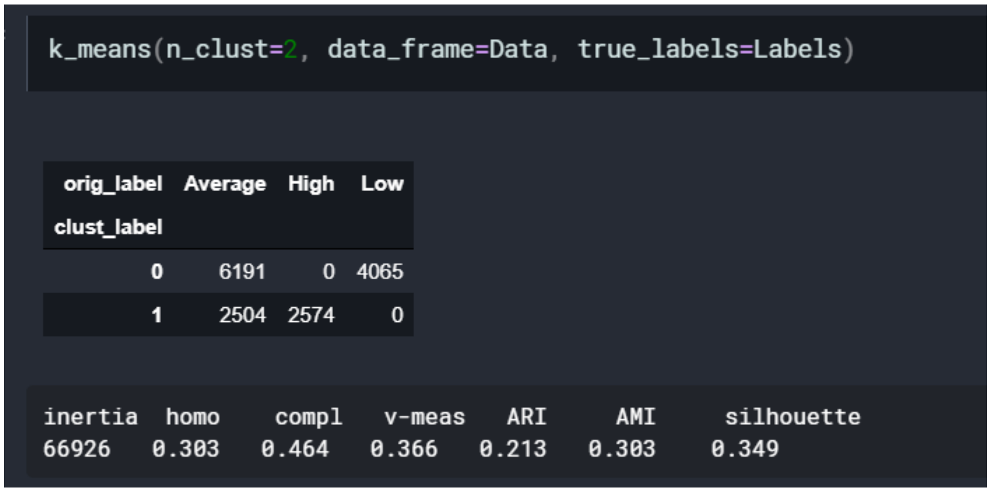
```
Labels = Data['plagiarism']
Data = Data.drop(['filenames', 'plagiarism'], axis = 1)
Labels_keys = Labels.unique().tolist()
Labels = np.array(Labels)
print('Plagiarism levels: ' + str(Labels_keys))
```

The labels will print the plagiarism levels as high, medium and low. The data is then scaled and standardized using scalar.fit_transform(data). The optimal cluster 'k' values are checked using the elbow method and the desired number is calculated (K=2 in this case). K-means function is computed in the cellblocks and the related scores such as inertia, v-meas, homo, ARI, AMI and silhouette score are outputted for the given data.

```
def k_means(n_clust, data_frame, true_labels):
 k_means = KMeans(n_clusters = n_clust, random_state=123, n_init=30)
   k_means.fit(data_frame)
   c_labels = k_means.labels_
```

```
df = pd.DataFrame({'clust_label': c_labels, 'orig_label': true_labels.tolist()})
ct = pd.crosstab(df['clust_label'], df['orig_label'])
y_clust = k_means.predict(data_frame)
display(ct)
print('% 9s' % 'inertia  homo    compl   v-meas   ARI     AMI     silhouette')
print('%i   %.3f   %.3f   %.3f   %.3f   %.3f    %.3f'
  %(k_means.inertia_,
  homogeneity_score(true_labels, y_clust),
  completeness_score(true_labels, y_clust),
  v_measure_score(true_labels, y_clust),
  adjusted_rand_score(true_labels, y_clust),
  adjusted_mutual_info_score(true_labels, y_clust),
  silhouette_score(data_frame, y_clust, metric='euclidean')))
```

With k=2 clusters, the silhouette score observed for the system data is 0.349 and 0.21 for k=3.



```
k_means(n_clust=2, data_frame=Data, true_labels=Labels)
```

orig_label	Average	High	Low
clust_label			
0	6191	0	4065
1	2504	2574	0

```
inertia  homo    compl   v-meas   ARI     AMI     silhouette
66926    0.303   0.464   0.366    0.213   0.303   0.349
```

Figure 34: K-means output for k=2 with related scores

Referring to figure 34, Upon giving the SimDec systems data to the k-means unsupervised clustering algorithm, it is observed that the optimal number of clusters for the operation would be 2 (k=2) for 15,344 records. The k-means performed in python clusters the data into two groups where ‘Average’ and ‘High’ labelled data (Label is dropped as shown in the python code above) is allocated to the second cluster and ‘Low’ and ‘Average’ is allocated to the first cluster. As you can see from the figure, the highest number of records are labelled as ‘Average’ are 6191 and therefore fluctuate in both the clusters. According to the theory of the algorithm, unsupervised k-means is performing well in terms of clustering with our systems data. Silhouette score ranges from -1 to 1 and any value closer to ‘1’ means that cluster is well separated from each other and value nearby ‘0’ denotes that clusters are overlapping. The silhouette scores obtained with our data (0.349) indicates that clusters are overlapping and the fact that records with ‘Average’ label tops the count supports the overlapping theory.

K-means with ‘R’ and ‘R-Studio’

The perfect explanation of k-means algorithm with visually represented clusters can be observed with ‘R’ programming language and R-Studio. ‘R’ and ‘Python’ programming languages have been closely associated with data analytics and machine learning, but when it comes to supervised classification / prediction, Python is more effective than ‘R’. ‘R’ is popular for providing interactive visualization packages that support the infographics and unsupervised clustering is all about diagrammatical representations. The packages which should be installed to perform the k-means clustering in R-studio are ‘cluster’, ‘devtools’, ‘factoextra’ and other that support visualization such as ‘ggplot2’ and ‘dplyr’. Following are the steps to perform k-means in R-studio:

1. Read csv file into R-studio, empty the ‘plagiarism’ column OR nullify it

R-code:

```
twentyKdata <- read.csv("C:/Users/batma/OneDrive/Desktop/twentyKdata.csv")
View(twentyKdata)
twentyKdata <- data.frame(twentyKdata[,-1], row.names = twentyKdata[,1])
twentyKdata[,c("plagiarism")] <- list(NULL)
twentyKdata <- scale(twentyKdata)
head(twentyKdata, n = 3)
```

2. Set seed to 123 and perform the clustering for ‘500’ samples. Including thousands of samples can cause abrupt visualizations and unclear analytics. Scale the sampled data with Euclidean distance and with Pearson coefficient as well.

R-code:

```
set.seed(123)
samset <- sample(1:3571, 500)
df <- twentyKdata[samset,]
```

```
df.scaled <- scale(df)
dist.eucl <- dist(df.scaled, method = "euclidean")
round(as.matrix(dist.eucl)[1:3, 1:3], 1)
library("factoextra")
dist.cor <- get_dist(df.scaled, method = "pearson")
df
view(df)
```

3. Load ‘factoextra’ library and visualize the Euclidean distance for all the values present in the variable. Find out the optimal number of clusters for the sample set of 500 records within the large data

R-code : fviz_nbclust(df, kmeans, method = "wss") + geom_vline(xintercept = 4, linetype = 2)

Output:

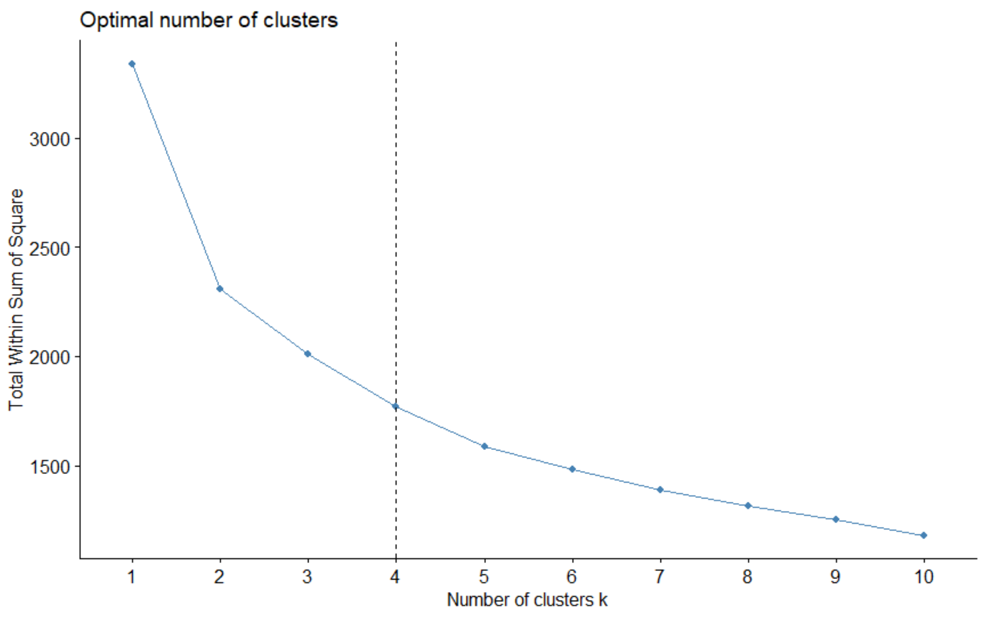


Figure 35: K-means: Optimal Number of clusters

4. Print the values with clusters for k=4 with 25 shuffles and list out the available components such as cluster, centers, totss, withinss, betweenss, size, iter and ifault

R code:

```
km.res <- kmeans(df, 4, nstart = 25)
print(km.res)
```

Output:

```
student1547.cstudent5413.c student1477.cstudent7735.c student1477.cstudent4804.c
student1477.cstudent1571.c

                  2                   1                   1                   1

student1547.cstudent1738.c student1571.cstudent2234.c student1502.cstudent3442.c
student1547.cstudent1725.c

                  3                   4                   2                   3

student1477.cstudent4163.c student1502.cstudent8069.c student1542.cstudent8187.c
student1483.cstudent4185.c

                  1                   3                   1                   3


Within cluster sum of squares by cluster:

[1] 407.6792 423.1623 501.9302 439.0249

 (between_SS / total_SS =  46.9 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"    "size"

[8] "iter"         "ifault"
```

5. Print the size and centers of the cluster and visualize the k-means clusters for 500 samples

R code:

```
km.res$size
km.res$centers
fviz_cluster(km.res, data = df,
palette = c("#2E9FDF", "#00AFBB", "#E7B800", "#FC4E07"),
ellipse.type = "euclid", # Concentration ellipse
star.plot = TRUE, # Add segments from centroids to items
repel = TRUE, # Avoid label overplotting (slow)
ggtheme = theme_minimal())
```

Output:

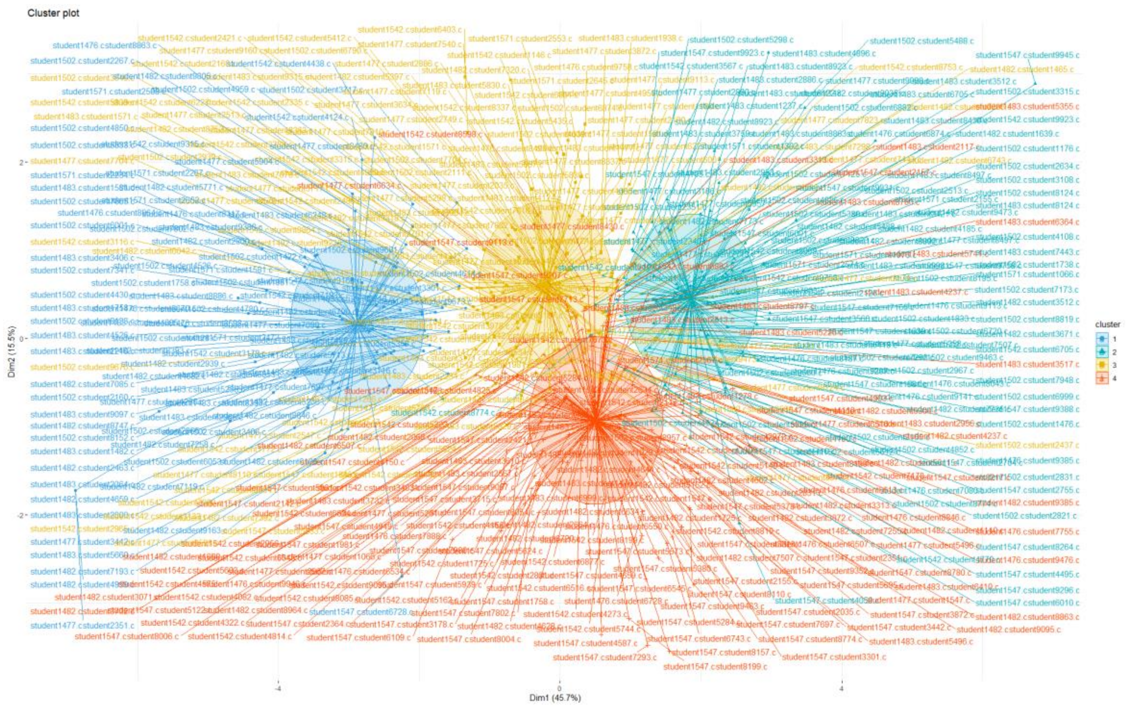


Figure 36: K-means Clusters for 500 Observations

As shown in figure 36, the four clusters are visible but the name conventions appear to be congested. The congestion is mainly because of the long names and the only way to purify it is to reduce the number of samples and try again with 100 or 200 samples. The best possible scenario to cluster and visualize 500 or more sample would be to use the CLARA clustering technique. CLARA stands for clustering large applications, which is an extension to the k-medoids variation, PAM. To reduce the memory and computation time, CLARA approach could be considered for large data. Without diving into command details, the cluster visualization obtained with CLARA is shown below as figure 37. The main intention behind performing the experimentation in 'R' and 'R-Studio' was to check if unsupervised clustering on SimDec system data could be visualized properly or not. Python libraries such as seaborn and matplotlib are incapable of plotting clusters for large datasets and therefore, there was a need to check the clustering visualization on another platform. As shown in figure 36, k-means clustering for 500 records appears to be congested whereas the clustering for large data seems pleasant and clear with CLARA clustering technique.

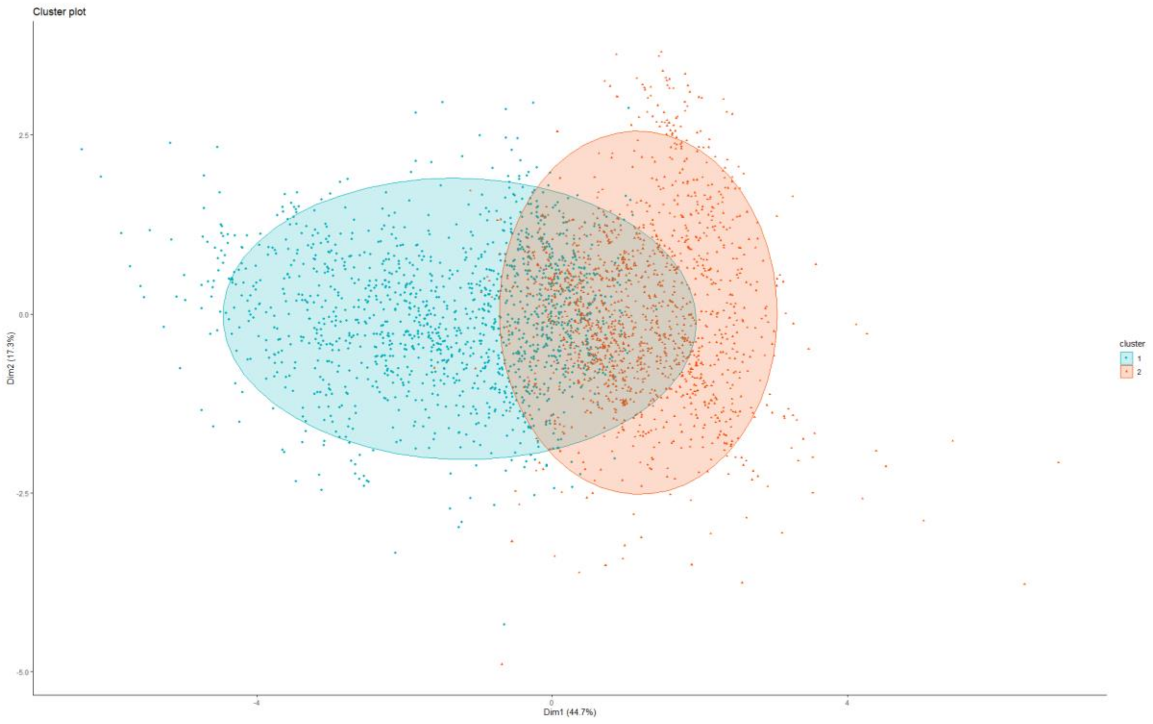


Figure 37: CLARA cluster visualization for 500 samples

5.3.2 Principal Component Analysis

Principal component analysis (PCA) is another clustering technique that acts as a dimensionality reduction mechanism and invented by Karl Pearson in 1901 [41]. PCA technique could be used to reduce a large set of variable into a small set that still contains most of the information in the large set. PCA for the same dataset as used for k-means has been conducted on the python environment. The first step is to find the optimal number of features for dimensionality reduction.

```
pca = PCA(random_state=123)
pca.fit(Data)
features = range(pca.n_components_)
plt.figure(figsize=(8,4))
plt.bar(features[:15], pca.explained_variance_[:15], color='lightskyblue')
plt.xlabel('PCA feature')
plt.ylabel('Variance')
plt.xticks(features[:15])
plt.show()
```

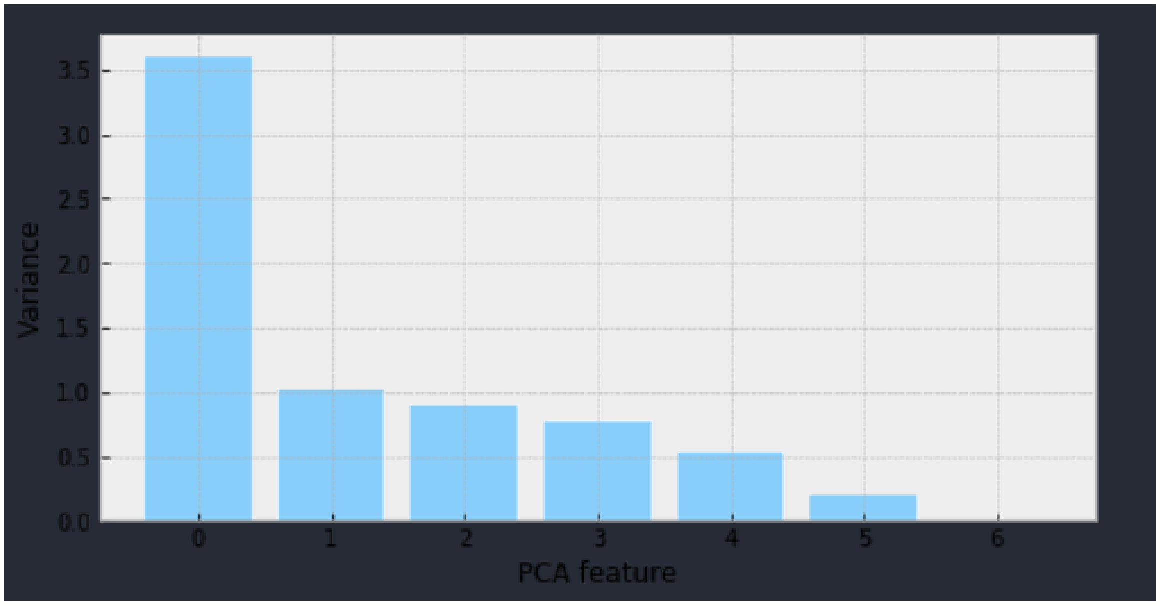
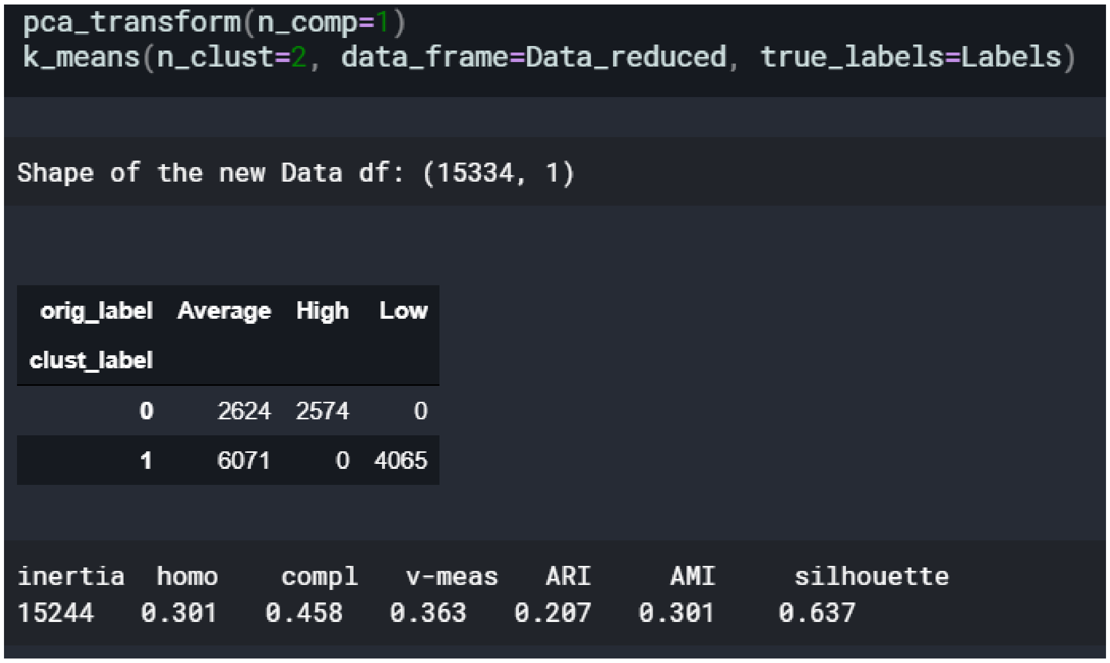


Figure 38: PCA features X variance for system data

Looking at the figure given above, 1-feature seems to be the best fit for our data as ‘0’ is default set to the maximum variance. If the PCA feature is equal to 1 and clusters = 2, the resulting components, including the silhouette score, are better than those of k-means.



```
pca_transform(n_comp=1)
k_means(n_clust=2, data_frame=Data_reduced, true_labels=Labels)
```

```
Shape of the new Data df: (15334, 1)
```

orig_label	Average	High	Low
clust_label			
0	2624	2574	0
1	6071	0	4065

```
inertia   homo    compl   v-meas   ARI     AMI     silhouette
15244    0.301   0.458   0.363   0.207   0.301    0.637
```

Figure 39: PCA transformation with two features

If we switch the PCA features from ‘1’ to ‘2’, the silhouette and the corresponding scores will decrease as the number of PCA components/features increase. From the unsupervised clustering study, it is clear that k-means perform exceptionally well for large data combined with the PCA technique.

5.4 Experimentation Analysis & Results

To conclude the second module of research, we will discuss the results and advantages/disadvantages of performing supervised and unsupervised analysis on the SimDec system data. To describe the dataset selected for experimentation under this module, the data table shown in figure 23 is concatenated with a class label or target variable as stated in table 8 in chapter 3 of this thesis. The dataset is typically designed for supervised learning as it contains a class label. Still, it could be used for unsupervised learning as all columns are numerical and discard the class label column. The three algorithms used for this study were logistic regression, multi-class SVM, and simple neural networks to proceed with the results discussion for supervised learning methods. The logistic regression was performed on the SimDec data consisting of around 18K records and a class label with categories such as high, average and low for plagiarism levels. The records indicate the information obtained from a pair of student assignment under comparison. The data was split into 80:20 ratios and the model was trained with three categories in the class label column. The precision, recall, f1-score and support obtained with logistic regression were 0.96, 0.98, 0.97 and 549. Due to the dataset's simple schema, the accuracy obtained with logistic regression was 0.98 or 98% with less than 60 false positives. The confusion matrix obtained for logistic regression is shown below as follows:

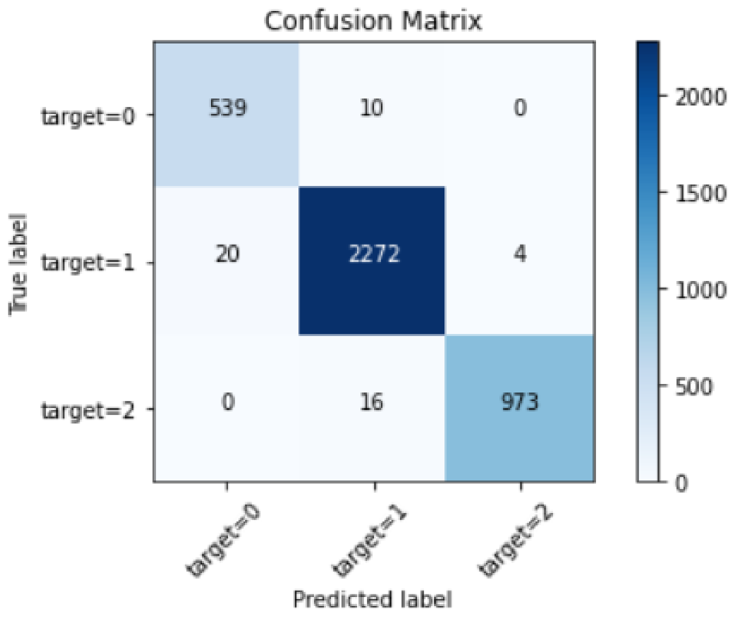


Figure 40: Logistic Regression Confusion Matrix

Multi-class SVM was trained on the same data in this comparative study with 6 numeric features. The predictor set was normalized for SVM training at the first and the library used to build the ensemble model was random forest classifier. The hyperparameter tuning using grid search and cross validation involved ‘rbf’ and ‘linear’ kernels with varying C from 0 to 1000. The datasets for training and testing were provided as separate files to the SVM and the scores obtained were 0.99 and 0.99 for training and testing respectively. SVM performed better than logistic regression at classification in the aspects of training and model evaluation. The last algorithm to be discussed for supervised learning is simple neural network algorithm. As mentioned in the solo subsection for neural networks, the model's architecture is sequential with three ‘ReLU’ activation layers and one softmax. The model is compiled with a loss function such as' Adam ', such as categorical cross-entropy and optimizer. The model is trained on 10 epochs with batch size of 20 and evaluated for the test data. The testing accuracy obtained with neural networks is 93.96% and could increase if number of epochs are increased with varying batch size and different optimizers. The neural networks model was saved, loaded back and tested on a different type of data without the class label. The predictions for the new data were reasonable and paved the way of success for this research module. All three supervised techniques perform exceptionally well of the system data and would work fine for the new real-world generated data. This implementation of ML supervised algorithms with SimDec system could reduce the time complexity and number of computations as the system won't be required to generate the class label. Unsupervised techniques help cluster and visualize the student assignments in groups but are less efficient when it comes to classification or time complexity reduction. The clustering visualization are not cumbersome to display on the web interface and is a good-to-have feature with the similarity detector system. Nonetheless, if it comes to decide one out of the two ML experimentations, supervised learning algorithms are recommended to collaborate the SimDec system as they can predict the class label for a student assignment record without the need of the system to calculate the class label based on some criteria, indirectly resulting in reduced time complexity. To summarize this chapter, the accuracies obtained from all the supervised learning algorithms are shown in the table given below:

Table 11: Accuracies of all supervised learning algorithms

Supervised Algorithms	Training Accuracy	Testing Accuracy
Logistic Regression	98%	98%
Support Vector Machine	99%	99%
Neural Networks	93%	95%

The next chapter would be the end of this thesis as all the experimentation has been conducted and results are available as shown in the all the other chapters above. Chapter 6 will conclude this thesis research with a short conclusion highlighting important points and a wide future scope for our proposed 'SimDec' system.

Chapter 6. Conclusions and Future Work

6.1 Conclusion

In any academic institution, several students submit their assignments electronically and the primary concern here is the e-plagiarism detection in the student assignments. Given the current COVID-19 situation where all the courses are being delivered online, students are asked to submit the practical assignments electronically. The probability of copying and plagiarizing assignments has increased heavily as there is not much of personal monitoring involved by the teachers. An act of submitting / copying someone else's work is considered as 'e-identity theft' or 'Plagiarism', which disobeys university dishonesty regulations and could lead to suspension or detention. The existing similarity detection tools use inefficient approaches such as attribute counting metrics (ATM) with the tokenization approach that involves the longest common substring (LCS) search method. Many similarity detector engines prefer using hashing techniques and syntax tree/AST modifiers for file matching if the focus is on the line-word comparison. YAP3, JPlag and MOSS are the current tools being used by many institutions and it is a complicated decision to make when it comes to recommending a tool above all others. Few notable disadvantages of these similarity detectors are lack of visual support (GUI), batch file processing, and a robust assistant tool. The similarity detection engine proposed in this thesis research addresses the challenges the evaluators and examiners face at professional institutes where students upload their assignments digitally. The designed 'SimDec' system follows a systematic ATM alongside a tokenizer (ANTLR) driven mainframe controlling system delivering lexical analysis computation with multiple similarity measures. The experimentation has been performed on the IEEE dataset consisting of 'C' and 'C++' corpora. The similarity measures considered for this experimentation include cosine similarity, n-grams, Levenshtein distance, Jaro & Jaro-Winkler and coefficients such as Dice, Jaccard, and F-1. The research is divided into two modules, the first one being the similarity detection process and the second one involves of machine learning classification and clustering. The data extracted from the source codes such as token scores and other data is stored in the relational database and the same data is given to the machine learning module of this research. According to the results and experimentation illustrated in chapter 5, supervised learning is more favorable and reliable than unsupervised because of smart prediction and ML indicators to detect the student assignment pair's plagiarism level. Adding a novelty feature to this implementation over the other existing software’s, a web application has been developed to represent analysis and visualization conducted throughout the procedure.

6.2 Future Work

The current research can be expanded by extending the detection process to the next level, which is syntactic analysis. The construction of a parser tree using ANTLR for one source code is complex and therefore will be more difficult to do the same for a bunch of files in a parallel processing environment. The expansion will improve the comparison accuracy as the source code controls, and constructs will be evaluated. Various parse tree algorithms for recursive descent parser and LR/LL can be used for similarity detection. The grammar for other programming languages such as Java, Python, COBOL, PASCAL, LISP etc can be included with the ANTLR tokenizer and the systems code could be modified accordingly. Adding this feature might increase the scope of the system and utilization in multiple applications. The current machine learning module of this thesis states the importance of supervised learning for predicting plagiarism levels for student assignments and reducing the time complexity of the system. In future, the ML techniques could be encapsulated in the systems archive and perform computations without the need of exporting data from MySQL back-end. Diagrammatic representations of critical analysis and insights within the comparison process will be essential and play the key role in this kind of research.

References

[1] Vedran Ljubovic. (2020). Programming Homework Dataset for Plagiarism Detection. *IEEE Dataport*. http://dx.doi.org/10.21227/71fw-ss32

[2] Matija Novak, Mike Joy, Dragutin Kermek. (2019). Source-code Similarity Detection Tools used in Academia: A systematic Review, *ACM 2019*

[3] Daniel Heres. (2017). Source Code Plagiarism Detection Using Machine Learning, *Utrecht University*.

[4] J. A. W. Faidhi and S. K. Robinson. 1987. An empirical approach for detecting program similarity and plagiarism within a university programming environment. Comput. Educ. 11, 1 (Jan. 1987), 11–19. DOI:https://doi.org/10.1016/0360-1315(87)90042-X

[5] Al-Khanjari, Zuhoor & Fiaidhi, & Al-Hinai, & Kutti, N.S. (2010). PlagDetect: A Java Programming Plagiarism Detection Plug-in. *ACM Inroads magazine*.

[6] Gitchell, David & Tran, Nicholas. (1999). Sim: A utility for detecting similarity in computer programs. *SIGCSE Bulletin* (Association for Computing Machinery, Special Interest Group on Computer Science Education). 31. 266-270. 10.1145/299649.299783.

[7] Pike R, Loki (2002) The sherlock plagiarism detector. http://www.cs.usyd.edu.au/~scilect/sherlock/, accessed date: 14 February 2016

[8] Prechelt, Lutz & Malpohl, Guido. (2003). Finding Plagiarisms among a Set of Programs with JPlag. *Journal of Universal Computer Science*. 8.

[9] Kamiya T, Kusumoto S, Inoue K (2002) CCFInder: amultilinguistic token-based code clone detection system for large scale source code*. Trans Softw Eng* 28(7):654–670

[10] Wise, Michael. (1996). YAP3: Improved Detection of Similarities in Computer Program and Other Texts. *ACM SIGCSE Bulletin*. 28. 10.1145/236452.236525.

[11] Ahtiainen A, Surakka S, Rahikainen M (2006) Plaggie: GNU-licensed source code plagiarism detection engine for Java exercises. In: *Baltic sea* '06, pp 141–142

[12] Jiang, Lingxiao & Misherghi, Ghassan & Su, Zhendong & Glondu, Stephane. (2007). DECKARD: scalable and accurate tree-based detection of code clones. *96-105. 10.1109/ICSE.2007.30.*

[13] Roy CK, Cordy JR (2008) NICAD: Accurate detection of near-miss intentional clones using flexible pretty printing and code normalization. In*: ICPC'08*, pp 172–181

[14] G¨ode N, Koschke R (2009) Incremental clone detection. In: *CSMR'09*, pp 219–228

[15] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P,Weiss R,Dubourg V et al (2011) *Scikit-learn: machine learning in python. J Mach Learn Res 12*(Oct):2825–2830.

[16] Cohen A (2011) Fuzzywuzzy: Fuzzy string matching in python. http://chairnerd.seatgeek.com/ fuzzywuzzy-fuzzy-string-matching-in-python/, accessed date: 14 March 2016

[17] Poulter G (2012) Python ngram 3.3. https://pythonhosted.org/ngram/, accessed date: 14 February 2016

[18] Harris S (2015) Simian – similarity analyser, version 2.4. http://www.harukizaemon.com/simian/, accessed date: 14 February 2016

[19] Python Software Foundation (2016) Difflib – helpers for computing deltas. http://docs.python.org/2/library/ difflib.html, accessed date: 14 February 2016

[20] Turk J, Stephens M (2016) A python library for doing approximate and phonetic matching of strings. https:// github.com/jamesturk/jellyfish, accessed date: 14 February 2016

[21] Cilibrasi R, Vitanyi PMB (2005) Clustering by compression. *Trans Inf Theory* 51(4):1523–1545

[22] Schleimer, Saul & Wilkerson, Daniel & Aiken, Alex. (2003). Winnowing Local Algorithms for Document Fingerprinting. *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 10. 10.1145/872757.872770.

[23] Vedran Ljubovic, “Programming Homework Dataset for Plagiarism Detection”, IEEE Dataport, 2020. [Online]. Available: http://dx.doi.org/10.21227/71fw-ss32. Accessed.Aug.17.2020

[24] V. Ljubovic and E. Pajic. (2020). Plagiarism Detection in Computer Programming Using Feature Extraction From Ultra-Fine-Grained Repositories. *IEEE Access, vol. 8, pp. 96505-96514*, 2020, doi: 10.1109/ACCESS.2020.2996146.

[25] Selfa, Diana & Carrillo, Maya & Boone, Ma. (2006). A Database and Web Application Based on MVC Architecture. 48 - 48.

[26] X. Li and N. Liu, "Research on L-MVC Framework. (2016) 17th International Conference on Parallel and Distributed Computing, *Applications and Technologies (PDCAT), Guangzhou, 2016*, pp. 151-154, doi: 10.1109/PDCAT.2016.043.

[27] Kembang Hapsari, Rinci & Azinar, Azmuri & Sugiyanto, Sugiyanto. (2017). Architecture Application Model View Controller (MVC) in Designing Information System of MSME Financial Report. *Quest Journals Journal of Software Engineering and Simulation.* 3. 36-41. 10.17605/osf.io/3z9r7.

[28] Levenshtein, V. I. (1996). “Binary Codes Capable of Correcting Deletions, Insertions and Reversals”, *Soviet Physics Doklady*, vol. 10, p. 707.

[29] Matthew A. Jaro (1989) Advances in Record-Linkage Methodology as Applied to Matching the 1985 Census of Tampa, Florida, *Journal of the American Statistical Association,* 84:406, 414-420, DOI: 10.1080/01621459.1989.10478785

[30] Winkler, W. E. (2006). "Overview of Record Linkage and Current Research Directions" (PDF*)*. Research Report Series, RRS.

[31] Sørensen, T. (1948). "A method of establishing groups of equal amplitude in plant sociology based on similarity of species and its application to analyses of the vegetation on Danish commons". *Kongelige Danske Videnskabernes Selskab*. **5** (4): 1–34.

[32] Dice, Lee R. (1945). "Measures of the Amount of Ecologic Association between Species". *Ecology*. **26** (3): 297–302. doi:10.2307/1932409. JSTOR 1932409.

[33] Gunawan, Dani & Sembiring, C & Budiman, Mohammad. (2018). The Implementation of Cosine Similarity to Calculate Text Relevance between Two Documents. *Journal of Physics: Conference Series.* 978. 012120. 10.1088/1742-6596/978/1/012120.

[34] David Maier. (1978). The Complexity of Some Problems on Subsequences and Supersequences. J. *ACM 25, 2* (April 1978), 322–336. DOI:https://doi.org/10.1145/322063.322075

[35] Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20, 273-297.

[36] Ben-Hur, Asa; Horn, David; Siegelmann, Hava; Vapnik, Vladimir N. (2001) . ""Support vector clustering" (2001);". *Journal of Machine Learning Research*. 2: 125–137.

[37] Cox, D. (1958). The Regression Analysis of Binary Sequences. *Journal of the Royal Statistical Society. Series B (Methodological), 20*(2), 215-242. Retrieved November 18, 2020, from http://www.jstor.org/stable/2983890

[38] McCulloch, W.S., Pitts, W. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics* 5, 115–133 (1943). https://doi.org/10.1007/BF02478259

[39] MacQueen, J. (1967) . Some methods for classification and analysis of multivariate observations. Proc. Fifth Berkeley Sympos. *Math. Statist. and Probability* (Berkeley, Calif., 1965/66) Vol. I: Statistics, pp. 281–297 Univ. California Press, Berkeley, Calif.

[40] Singh, Dilpreet & Reddy, Chandan. (2014). A survey on platforms for big data analytics. Journal of Big Data. 2. 10.1186/s40537-014-0008-6.

[41]

Footnotes

[1]Antonio Meucci, “Online Plagiarism Checking,” PlagScan, 2009, https://www.plagscan.com/en/.

[2]Alon Yamin, “Plagiarism Detector: AI Based Anti-Plagiarism Online,” Copyleaks, 2015, https://copyleaks.com/.

[3]IT Company Phase One Karma, “Plagiarism Checker for Educators and Students,” Unicheck, 2014, https://unicheck.com/.

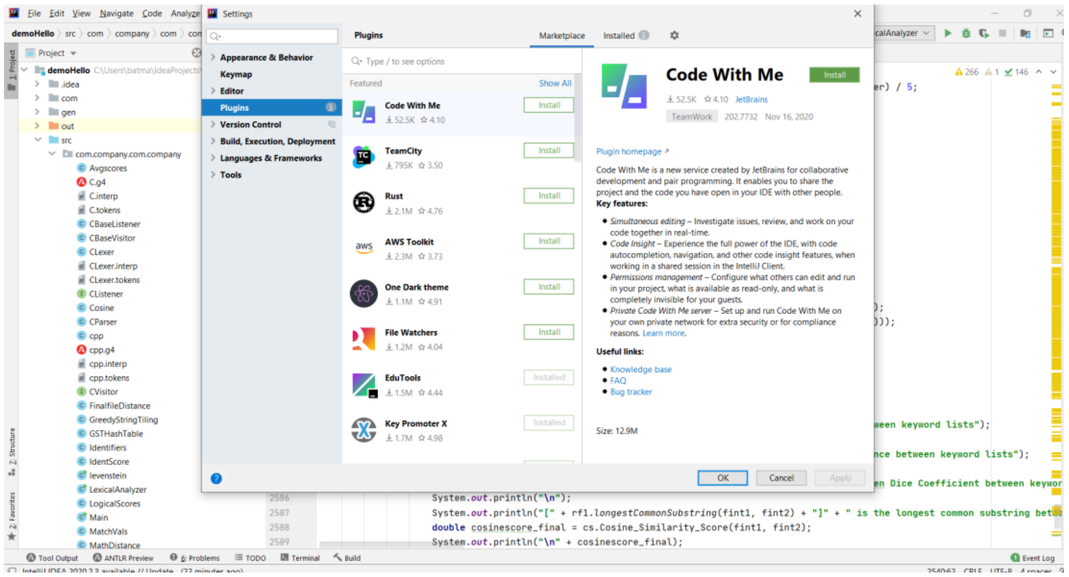
[4]Anonymous Author, “Code with Integrity,” Codequiry, accessed November 29, 2020, https://codequiry.com/.

[5]Vedran Ljubovic, “Programming Homework Dataset for Plagiarism Detection,” IEEE DataPort (IEEE, May 8, 2020), https://ieee-dataport.org/open-access/programming-homework-dataset-plagiarism-detection.

Appendix A.

Installation of ANTLR & Other libraries

1. Open Intellij IDEA and navigate to settings → plugins



Code With Me

52.5K 4.10 JetBrains

TeamWork 202.7732 Nov 16, 2020

Plugin homepage

Code With Me is a new service created by JetBrains for collaborative development and pair programming. It enables you to share the project and the code you have open in your IDE with other people.
Key features:

- *Simultaneous editing* – Investigate issues, review, and work on your code together in real-time.
- *Code Insight* – Experience the full power of the IDE, with code autocompletion, navigation, and other code insight features, when working in a shared session in the IntelliJ Client.
- *Permissions management* – Configure what others can edit and run in your project, what is available as read-only, and what is completely invisible for your guests.
- *Private Code With Me server* – Set up and run Code With Me on your own private network for extra security or for compliance reasons. Learn more.

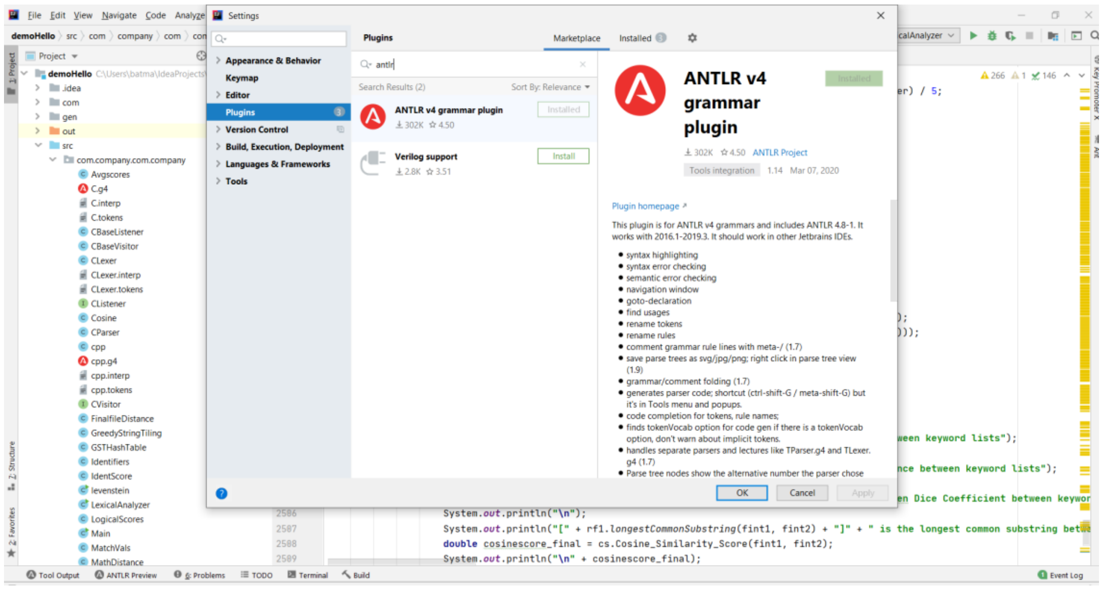
Useful links:

- Knowledge base
- FAQ
- Bug tracker

Size: 12.9M

```
System.out.println("\n");
System.out.println("[" + rf1.longestCommonSubstring(fint1, fint2) + "]" + " is the longest common substring bet
double cosinescore_final = cs.Cosine_Similarity_Score(fint1, fint2);
System.out.println("\n" + cosinescore_final);
```

2. Type ‘ANTLR’ in the search bar and install the plug-in for direct add-on to the environment



ANTLR v4 grammar plugin

302K 4.50 ANTLR Project

Tools integration 1.14 Mar 07, 2020

Plugin homepage

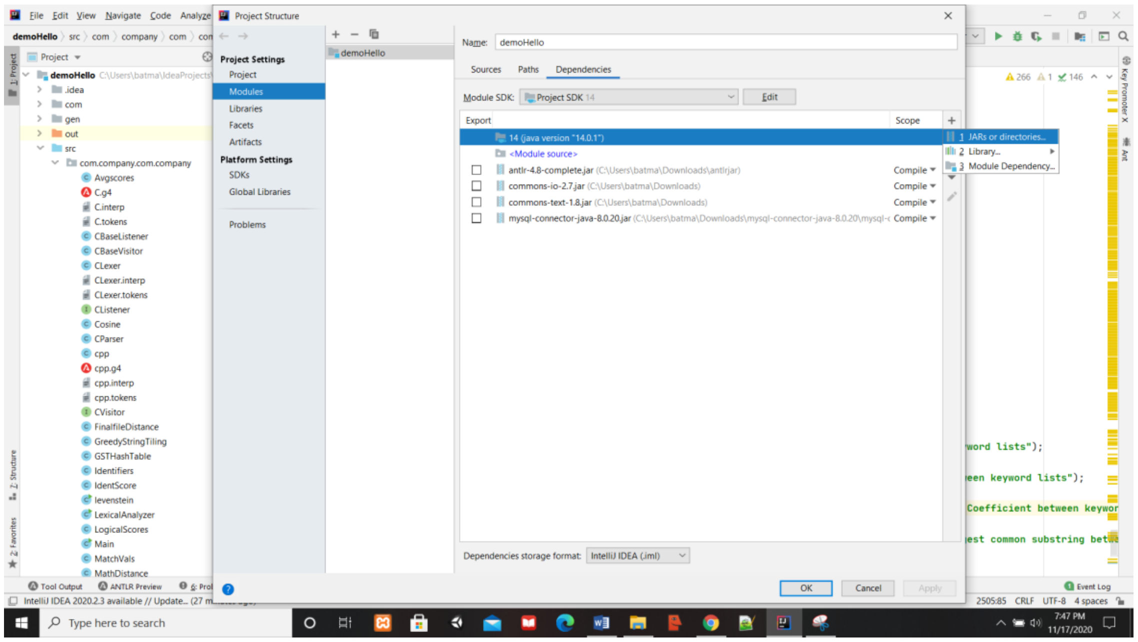
This plugin is for ANTLR v4 grammars and includes ANTLR 4.8-1. It works with 2016.1-2019.3. It should work in other Jetbrains IDEs.

- syntax highlighting
- syntax error checking
- semantic error checking
- navigation window
- goto-declaration
- find usages
- rename tokens
- rename rules
- comment grammar rule lines with meta-/ (1.7)
- save parse trees as svg/jpg/png; right click in parse tree view (1.9)
- grammar/comment folding (1.7)
- generates parser code; shortcut (ctrl-shift-G / meta-shift-G) but it's in Tools menu and popups.
- code completion for tokens, rule names;
- finds tokenVocab option for code gen if there is a tokenVocab option, don't warn about implicit tokens.
- handles separate parsers and lectures like TParser.g4 and TLexer.g4 (1.7)
- Parse tree nodes show the alternative number the parser chose

```
System.out.println("\n");
System.out.println("[" + rf1.longestCommonSubstring(fint1, fint2) + "]" + " is the longest common substring bet
double cosinescore_final = cs.Cosine_Similarity_Score(fint1, fint2);
System.out.println("\n" + cosinescore_final);
```

3. Download MySQL connector from https://dev.mysql.com/downloads/connector/j/ and add the jar file to the IDE via external libraries.

4. Go to File → Project Structure → Modules → Dependencies → Add new jar files



Name: demoHello

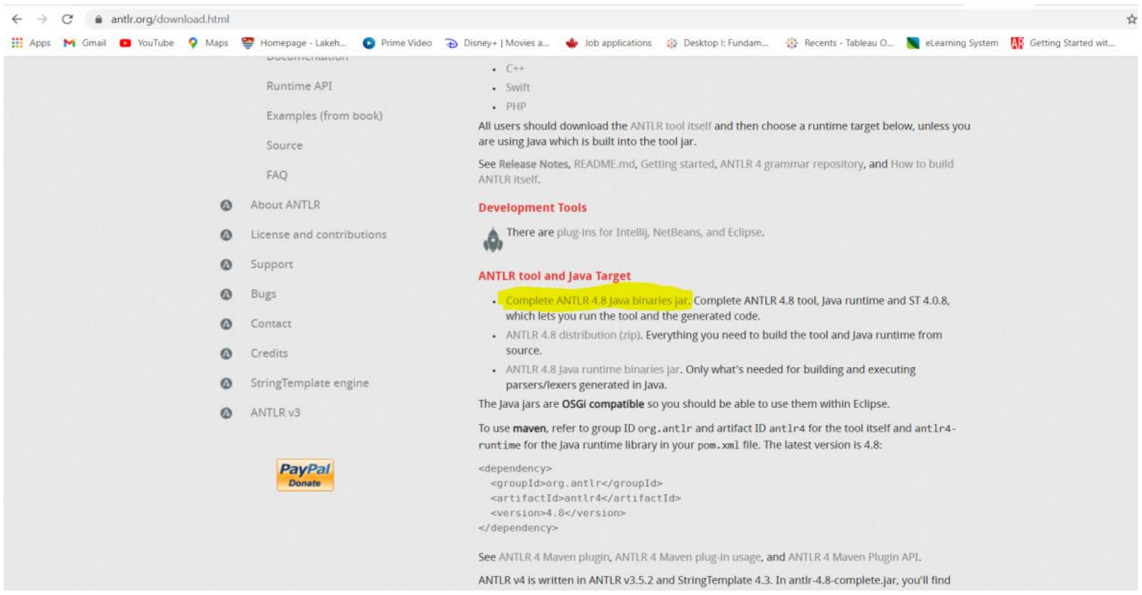
Module SDK: Project SDK 14

Export		Scope
	14 (java version "14.0.1")	
	<Module source>	
☐	antlr-4.8-complete.jar (C:\Users\batma\Downloads\antlrjar)	Compile
☐	commons-io-2.7.jar (C:\Users\batma\Downloads)	Compile
☐	commons-text-1.8.jar (C:\Users\batma\Downloads)	Compile
☐	mysql-connector-java-8.0.20.jar (C:\Users\batma\Downloads\mysql-connector-java-8.0.20\mysql-c	Compile

Dependencies storage format: IntelliJ IDEA (.iml)

5. Now that ANTLR plug-in is added, let's add ANTLR Complete jar file to the system as the final requirement by following step 4.

6. Download the jar file from ANTLR's parent site - https://www.antlr.org/download.html



- C++
- Swift
- PHP

All users should download the ANTLR tool itself and then choose a runtime target below, unless you are using Java which is built into the tool jar.

See Release Notes, README.md, Getting started, ANTLR 4 grammar repository, and How to build ANTLR itself.

Development Tools

There are plug-ins for Intellij, NetBeans, and Eclipse.

ANTLR tool and Java Target

- Complete ANTLR 4.8 Java binaries jar. Complete ANTLR 4.8 tool, Java runtime and ST 4.0.8, which lets you run the tool and the generated code.
- ANTLR 4.8 distribution (zip). Everything you need to build the tool and Java runtime from source.
- ANTLR 4.8 Java runtime binaries jar. Only what's needed for building and executing parsers/lexers generated in Java.

The Java jars are **OSGi compatible** so you should be able to use them within Eclipse.

To use **maven**, refer to group ID `org.antlr` and artifact ID `antlr4` for the tool itself and `antlr4-runtime` for the Java runtime library in your `pom.xml` file. The latest version is 4.8:

```
<dependency>
  <groupId>org.antlr</groupId>
  <artifactId>antlr4</artifactId>
  <version>4.8</version>
</dependency>
```

See ANTLR 4 Maven plugin, ANTLR 4 Maven plug-in usage, and ANTLR 4 Maven Plugin API.

ANTLR v4 is written in ANTLR v3.5.2 and StringTemplate 4.3. In antlr-4.8-complete.jar, you'll find

Appendix B. Lexical Analyzer: Java code Samples

1. System Code: Java for specifying folder path with one to one file comparison and ANTLR grammar selection code snippet

```
String path1 = "F:\\Lakehead
Subjects\\Thesis_summer_spring_fall2020\\newz1_cpp";

        File folder1 = new File(path1);
        System.out.println(folder1.listFiles());
        List<File> foldlist1 = Arrays.asList(folder1.listFiles());
        File[] filesList1 = folder1.listFiles();

        for(File f1: filesList1) {
            System.out.println(f1.getName());
            for(File f2: filesList1) {
                if(!(f1.getName().equals(f2.getName()))) {
                    //System.out.println(f2.getName());

                    FileInputStream reader = new FileInputStream(f1);
                    FileInputStream reader1 = new FileInputStream(f2);

System.out.println("============================================");
                    System.out.println(f1.getName());
                    System.out.println(f2.getName());

System.out.println("============================================");
                    ArrayList<String> lines = new ArrayList<>();
                    ArrayList<String> lines1 = new ArrayList<>();
                    Map<String, List<String>> symbolTable = new
HashMap<String, List<String>>();
                    Map<String, List<String>> symbolTable1 = new
HashMap<String, List<String>>();
                    //ystem.out.println("yeahhh");
                    //System.out.println(f1.getName().matches("\\s$"));

//System.out.println(f1.getName().substring(f1.getName().length() -
1));
                    //System.out.println(f1.getName().endsWith("c"));

                    if(f1.getName().endsWith("c") &&
f2.getName().endsWith("c"))
                    {

                        ANTLRInputStream input = new
ANTLRInputStream((reader));
                        ANTLRInputStream input1 = new
ANTLRInputStream((reader1));
                        CLexer lexer = new CLexer(input);
                        CLexer lexer1 = new CLexer(input1);
                        //CParser parser = new CParser(input);
                        Token token = lexer.nextToken();

                        while (token.getType() != CLexer.EOF) {
                            //System.out.println(token.getText());
                            token = lexer.nextToken();
                            List<String> list =
Arrays.asList(token.getText());
                            lines.addAll(list);
                        }

                        Token token1 = lexer1.nextToken();

                        while (token1.getType() != CLexer.EOF) {
                            //System.out.println(token.getText());
                            token1 = lexer1.nextToken();
                            List<String> list1 =
Arrays.asList(token1.getText());
                            lines1.addAll(list1);
                        }
                    }
                    else if(f1.getName().endsWith("cpp") &&
f2.getName().endsWith("cpp")) {

                        ANTLRInputStream inputcpp = new
ANTLRInputStream((reader));
                        ANTLRInputStream input1cpp = new
ANTLRInputStream((reader1));
                        cpp lexer = new cpp(inputcpp);
                        cpp lexer1 = new cpp(input1cpp);
                        //CParser parser = new CParser(input);

                        Token token = lexer.nextToken();

                        while (token.getType() != cpp.EOF) {
                            //System.out.println(token.getText());
                            token = lexer.nextToken();
                            List<String> list =
Arrays.asList(token.getText());
                            lines.addAll(list);
                        }

                        Token token1 = lexer1.nextToken();

                        while (token1.getType() != cpp.EOF) {
                            //System.out.println(token.getText());
                            token1 = lexer1.nextToken();
                            List<String> list1 =
Arrays.asList(token1.getText());
                            lines1.addAll(list1);
                        }
                    }
```

2. Java Code for MySQL Connection

```
try {
                        connect1 = DriverManager

.getConnection("jdbc:mysql://localhost:3306/demo?"
                                        + "user=root&password=root");
                        statement1 = connect1.createStatement();
                        preparedStatement1 = connect1
                                .prepareStatement("insert into
demo.scores values (?, ?, ?, ?, ?, ? , ?, ?)");
                        //preparedStatement1.setString(1,
newList.get(i).getName().concat(newList1.get(i).getName()));
                        preparedStatement1.setString(1,
f1.getName().concat(f2.getName()));
                        preparedStatement1.setLong(2, keyocc);
                        preparedStatement1.setLong(3, math1);
                        preparedStatement1.setLong(4, numcount1);
                        preparedStatement1.setLong(5, logcount);
                        preparedStatement1.setLong(6, opcount);
                        preparedStatement1.setLong(7, total);
                        preparedStatement1.setLong(8, dis);
                        preparedStatement1.executeUpdate();

                        preparedStatement1 = connect1
                                .prepareStatement("SELECT filenames,
keywordscore, mathopscore, numericalscore, logicalscore, otheropscore,
totalsimilarity, totaldissimilarity from demo.scores");
                        resultSet1 = preparedStatement1.executeQuery();
                        connect1.close();
                    } catch (SQLException e) {
                        // TODO Auto-generated catch block
                        System.out.println("error in db");
                        e.printStackTrace();
                    }
```