DOKUZ EYLÜL UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

# ANALYZING SOURCE CODES AND DETECTING SIMILARITIES

by

Fatma BOZYİĞİT

June, 2015

İZMİR

# ANALYZING SOURCE CODES AND DETECTING SIMILARITIES

**A Thesis Submitted to the**
**Graduate School of Natural and Applied Sciences of Dokuz Eylül University**
**In Partial Fulfillment of the Requirements for the Master of**
**Science in Computer Engineering**

**by**
**Fatma BOZYİĞİT**

**June, 2015**
**İZMİR**

## M.Sc THESIS EXAMINATION RESULT FORM

We have read the thesis entitled **"ANALYZING SOURCE CODES AND DETECTING SIMILARITIES"** completed by **FATMA BOZYİĞİT** under supervision of **PROF. DR. ALPKUT**and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

[illegible]

Prof. Dr. Alp KUT

Supervisor

[illegible]

Asst.Prof.Dr. Derya BİRANT

(Jury Member)

[illegible]

Y.Doç.Dr. Reyat Yilmaz

(Jury Member)

[illegible]

Prof.Dr. Ayşe OKUR
Director
Graduate School of Natural and Applied Sciences

# ACKNOWLEDGEMENTS

I would like to thank to my supervisor, Prof. Alp KUT, for his support, supervision and useful suggestions throughout this study. It has been a privilege to work with him.

I am deeply grateful to my advisor, Asst. Prof. Deniz KILINÇ for his detailed and constructive comments, and for his important support throughout this work.

Finally, I would like to offer my special thanks to my family for their continuous support and encouragement throughout my studies.

Fatma BOZYİĞİT

## ANALYZING SOURCE CODES AND DETECTING SIMILARITIES

## ABSTRACT

Plagiarism in academic institutions is often expressed as copying someone else’s work (i.e., another students or from sources such as books). By reason of the fact that innovations and developments are occurred in technology of late years, massive increase of software applications is monitored. Concordantly, plagiarism issue becomes more significant day by day. Plagiarism of programming source codes is an undesirable situation in the many fields of software development world. Especially in educational field, it is obviously realized that plagiarism in programming courses increases consistently. The aim of this study is attempting to answer questions such as “which codes are similar?”, “what similarity ratios are?” in order to prevent plagiarism among college students who attend programming courses.

There are many methods and tools are available to find similarities between program codes. Generally traditional methods are preferable while detecting similarities among source codes. One of these traditional methods is finding metrics in software documents. However, different approaches are seen in recent years while solving plagiarism problems. N-gram method that belongs Natural Language Process (NLP) can be given as example of different approaches.

While developing the proposed methodology, metric extraction method (fingerprint system), N-gram algorithm and Vector Space Model (VSM) were considered. Information Retrieval (IR) System and Cosine Normalization (CN) methods were utilized to calculate similarity ratios.

Experimental study was performed on datasets of two different kinds. First type was created by collecting assignments of students who attend programming courses in Software Engineering Department of Celal Bayar University. Second type is yielded by changing source code examples in different forms.  The results obtained provide convincing evidence that the study is fit the purpose. The experimental results about proposed methods give success when compared with the previous methods.

**Keywords:** Plagiarism source code, software metrics, n-gram, vector space model, cosine normalization.

**KAYNAK KODLARIN ANALİZİ VE BENZERLİK TESPİT EDİLMESİ**

## ÖZ

Akademik kurumlarda intihal konusu bir başkasının emeğinin çalınması olarak açıklanmaktadır. Son yıllarda teknolojide meydana gelen yenilikler ve gelişmeler sebebi ile yazılım uygulamaları miktarında oldukça büyük artış gözlenmektedir. Buna paralel olarak yazılım projelerinde çalıntı konusu önemli bir sorun haline gelmektedir. Bir programcı tarafından oluşturulmuş bir yazılımın başkaları tarafından intihali yazılım dünyasında birçok alanda istenmeyen durumdur. Bu çalışmada, özellikle eğitim alanında öğrenciler arasında kod paylaşımının önüne geçmek, ders değerlendirirken haksızlıkların oluşmasını engellemek amacı ile öğrenciler tarafından hazırlanmış olan yazılımların içerisinde hangilerinin benzerlik gösterdiği, benzerlik oranlarının ne olduğu gibi soruların cevabının bulunması hedeflenmektedir.

Kaynak kodlar içerisinde benzerlik bulma konusunda hali hazırda kullanılan birçok method ve araç vardır. Benzerlikleri ortaya çıkarma konusunda genellikle geleneksel yöntemler kullanılmaktadır. Bu geleneksel yöntemlerden biri yazılıma ait metriklerin çıkarımı ve belirlenmesidir. Bu yöntemle beraber son yıllarda farklı methotlar da ortaya çıkarılmıştır. Bunlardan birisi doğal dil işleme alanına dâhil olan N-gram algoritmasıdır.

Bu çalışma gerçekleştirilirken metrik çıkarma(parmak izi belirleme) yöntemi, N-gram algoritması, Vektör Uzay Modeli ve Kosinüs Normalizasyon yöntemlerinden faydalanılmıştır. Bilgi elde etme sistemi (IR) ve Kosinüs Normalizasyon yöntemi benzerlik oranlarını hesaplamak için kullanılmaktadır.

Deneysel çalışmalar iki farklı tip veri seti üzerinde yapılmıştır. İlk veri seti Celal Bayar Üniversitesi Yazılım Mühendisliği Bölümü öğrencilerine ait programlama ödevlerinden oluşmaktadır. İkinci veri seti ise belirli bir kodun farklılaştırılması ile elde edilmiş kaynak kodları içermektedir. Elde edilen sonuçlar çalışmanın amacına uygun bir şekilde gerçekleştirildiğini göstermektedir. Önerilen metodlara ait deneysel sonuçlar önceki metotlar ile karşılaştırıldığında başarılı sonuçlar vermektedir.

**Anahtar kelimeler:** Kaynak kod hırsızlığı, yazılım metrikleri, N-gram algoriması, vektör uzay modeli, kosinüs normalizasyonu.

**CONTENTS**

**Page**

## LIST OF FIGURES

**Page**

## LIST OF TABLES

**Page**

# CHAPTER ONE
# INTRODUCTION

## 1.1 General

Plagiarized code is a source code example which its source cannot be understood in detail most of the times. If a license of software allows using entire or some parts of source code, there is no problem while citing and using it. However, if citing or appropriating a source code is not allowed, this is out of line in terms of ethics. This issue is legally remarked in Intellectual and Artistic Works and while computer programs are included in the scope of works of science, the owner of source code is discussed as an author. Plagiarism of source code is an important problem that can be faced every time, in everywhere. For example, using a source code of a program which is developed specifically for a company without permission is a common plagiarism situation. Another example can be seen in education area. Especially, programming course instructors indicate that source code theft issues pose a major problem while evaluating students' projects and home-works. Accordingly, unfair distributions can appear in grading.

Based on continuing development of technology, applications in field of software increase correspondingly and plagiarism stands out as a big problem. There are many methods to understand whether code is stolen or not and to prevent code theft. One of these methods is evaluating a software tool which finds similarity ratios among source codes. Already developed tools are available and have been using in many fields such as education. For example, Plague, JPlag and YAP applications are well-known tools. Many instructors in universities use these applications in order to check whether the assignments in programming courses are copy or not.

The necessary steps to solve plagiarism problems on source codes are much harder than natural language processing (NLP). The traditional method is extracting source code metrics before similarity check. However, there are some disadvantages in this traditional approach. For example, software metrics are programming language dependent. Metrics which are created to specify characteristics of Java programming language may not be appropriate for C or Pascal. Another disadvantage is that the metric selection is not a trivial process and usually involves setting thresholds in order to eliminate metrics which aren't correlated to the classification model.

## 1.2 Purpose

The aim of this study is to find the similarity ratios among source codes belonging to a programming course and attempt to decide whether or not two or more programs are plagiarized. To carry out study, N-gram algorithm, Vector Space Model (VSM) and Information Retrieval (IR) system are utilized. Since N-gram algorithm is language independent and does not contain disadvantages of traditional methods, it has been selected in this study. Also, some metrics are extracted from the source code examples.

In the previous studies it is utilized only bi-gram and tri-gram methods to check the source codes are copy or not. In the thesis, bi-gram and tri-gram results are placed into Vector Space Model and similarity ratios are found. Also some metrics that give clue about the code writer are obtained. Furthermore, the fingerprint system is touched on. Some software metrics are specified and value of them calculated to scrutinize similarity results.

## 1.3 Organization of the Thesis

The rest of paper is organized as follows:

In Chapter 2, general information about related works and literature search about workings in source code plagiarism issue.

Chapter 3 details plagiarism and how students create copy codes from the original one.

Chapter 4 introduces the Text Mining and its methods in detail.

In Chapter 5, Fingerprint system and extracting metrics methods to find source code similarity are described generally.

In Chapter 6, the experimental datasets that have created for the application and experimental results are touched on.

Chapter 7 gives information about application which is developed for this study.

Chapter 8 concludes the paper and gives some information about future works.

# CHAPTER TWO
# RELATED WORKs

## 2.1 Literature Review

Joy and Luck dwell on plagiarism in assignments of programming courses. They explain plagiarism as out of favor making copy of documents or source codes. In the study, it is claimed if students in the programming course are in high number, detecting and controlling copies in assignments can be difficult. Also similarities among programs don’t refer to plagiarism all the time. In the study, it is inspected that how it can be decided about the code is copy or not. Source codes are divided into tokens that take value as name, operator, loop etc.. After filtering out unnecessary information, incremental comparison step is completed and similarity ratios among codes are obtained. In incremental comparison step, pair of programs is compared five times; in their original form, with the white spaces removed, with all comments removed etc… So they provide obtaining more consistent similarity results (Joy & Luck, 1999).

Jones indicates that plagiarism is an ethical problem can be faced always in the academic area. It is also mentioned that trying to detect copy codes in programming courses is so difficult for educators in terms of presenting proofs about copies, wasting time and emotional burden because of charging a student as cheater. Jones develop an application to give evidences of plagiarism to students. So, objection of students and arguing between instructors and students can be terminated owing to results of application. In metric based system, physical profiles that include general parameters such as number of lines, words and characters are created at first. Then, Helstead profiles that divide source code into tokens and store the frequencies of tokens, is evaluated. Lastly, two profiles are combined and distances of patterns of profiles from the each other computed (Jones, 2001).

Culwin and Thomas mention about plagiarism problem that increases in academic institutions. They elaborate why students steal the information and show it as their own while they do assignments. They perform a study to dissolve plagiarism. So, their study helps instructors to understand which assignment is copy. The study consist of four stage; collection, detection, confirmation and investigation. At collection stage, students use web form submissions, so collection of assignment is completed via Web. After collecting the data, detection stage is started and similarity ratios among documents are obtained. At confirmation step, instructors should check whether the similarity results are consistent. Because, two students whom similarity ratio is high, may use same web site while doing assignments, so they cannot be charged with plagiarism. The process is terminated with investigation step and students who will be punished are determined. It is briefly pointed that revealing proof of copy is so necessary to eliminate plagiarism in the study (Culwin & Thomas, 2001).

Frantzeskou indicates that to solve authorship disputes in software area, not only finding similarities among programs is enough, but also identifying source code authors is necessary. So, she developed SCAP Method to specify owner of source code. The author underscores that SCAP method is effective on all programming language. Also, it is claimed that SCAP Method can work with simple profile examples that include a few code lines and a few examples of profile is enough to get good results. In SCAP, after finding N-gram frequencies, Simplified Profile Intersection (SPI) value is counted. Value of SPI measures the intersection of source code documents and gives a similarity ratio (Frantzeskou, 2007).

Khreisat has used N-gram algorithm with machine learning approach in Arabic text classification study. After comparing N-gram results (1-gram, 2-gram etc.), tri-gram has been decided by the author. The base of study has been obtaining tri-gram frequencies. Their datasets contain Arabic text documents that was gathered from online Arabic newspapers. Firstly, the pre-processing step has been completed to distinguish punctuation marks, stop words from the documents. Then, N-gram algorithm step has been started. For each document tri-gram profiles has been generated for classification. The tri-gram results that belong each documents, has been stored in text files. The last step has been calculating Mannathan and Dice Measure for specifying the suitable category (Khreisat, 2008).

Cavnar and Trankle have preferred the N-gram method in their text classification work, due to provide fault tolerance in textual. Usenet News Group articles have been used while generating dataset. The corpus of working has depended on counting and comparing N-gram frequencies.  Firstly, the profiles has been created for each category, then distance from the profile for documents in dataset have been measured. Cavnar and Trankle have developed a simple distance measure system for counting differences of documents from profiles. They have called it as “out of place” measure. The distances that calculated with “out of place” measure have been ranged and documents that have minimum distances, have placed suitable categories (Cavnar and Trankle, 1994).

Study published by Keselj is one of the studies based N-gram algorithm. This study have inspected source code documents which has been written with Pascal programming language and have found similarities. After obtaining N-gram frequencies, frequencies with the highest value have been chosen and eliminated the other. Final step is counting Relative Distance value which is specific for Keselj’s study. The owner of the test document detected among the authors which has been identified previously (Keselj, 2007).

Krsul and Spafford developed a software analyzer program to automate the coding of software metrics. The software analyzer extracted layout, style and structure features from 88 C programs belonging to 29 known authors. A tool was developed to visualize the metrics collected and help select those metrics that exhibited little within-author variation, but large between-author variation. Discriminant function analysis was applied on the chosen subset of metrics to classify the programs by author. The experiment achieved 73% overall accuracy (Krsul & Spafford, 1995).

MacDonell and Gray have automated authorship identification of computer programs written in C++. Gray, Sallis and MacDonell 1998 developed a dictionary based system called IDENTIFIED (Integrated Dictionary-based Extraction of Non-language dependent Token Information for Forensic Identification, Examination, and Discrimination) to extract source code metrics for authorship analysis. In MacDonell and Gray's 2001 work, satisfactory results were obtained for C++ programs using case-based reasoning, feed-forward neural network, and multiple discriminant analysis. The best prediction accuracy – at 88% for 7 different authors-- was achieved using Case-Based Reasoning (MacDonell & Gray, 2001).

Focusing on Java source code, Ding and Samadzadeh investigated the extraction of a set of software metrics that could be used to identify the author. A set of 56 metrics of Java programs was proposed for authorship analysis. The contributions of the selected metrics to authorship identification were measured by canonical discriminant analysis. Forty-six groups of programs were diversely collected. They achieved a classification accuracy of 87.0% with the use of canonical variates (Ding & Samadzadeh, 2004).

## 2.2 Tools Developed for Finding Source Code Similarities

JPlag uses the same comparison algorithm as YAP3, but with optimised run time efficiency. In JPlag the similarity is calculated as the percentage of token strings covered. One of the problems of JPlag is that files must parse to be included in the comparison for plagiarism, and this causes similar files to be missed. Also, JPlag's user defined parameter of minimum-match length is set to a default number. Changing this number can alter the detection results (for better or worse) and to alter this number one may need an understanding of the algorithm behind JPlag (i.e RKR-GST). JPlag is implemented as a web service and contains a simple but efficient user interface. The user interface displays a list of similar file pairs and their degree of similarity, and a display for comparing the detected similar files by highlighting their matching blocks of source-code fragments.

Sherlock also implements a similar algorithm to YAP3. Sherlock converts programs into tokens and searches for sequences of lines (called runs) that are common in two files. Similarly to the YAP3 algorithm Sherlock searches for runs of maximum length. Sherlock's user interface displays a list of similar file pairs and their degree of similarity, and indicates their matching blocks of source-code fragments found within detected file pairs. In addition, Sherlock displays quick visualisation of results in the form of a graph where each vertex represents a single source-code file and each edge shows the degree of similarity between the two files. The graph only displays similarity (i.e., edges) between files above the given user defined threshold. One of the benefits of Sherlock is that, unlike JPlag, the files do not have to parse to be included in the comparison and there are no user defined parameters that can influence the system's performance. Sherlock, is an open-source tool and its token matching procedure is easily customisable to languages other than Java. Sherlock is a stand-alone tool and not a web-based service like JPlag and MOSS. A stand alone tool may be more preferable to academics with regards to checking student files for plagiarism when taking into consideration confidentiality issues.

MOSS (Measure of Software Similarity) is based on a string-matching algorithm that functions by dividing programs into k-grams, where a k-gram is a contiguous substring of length k. Each k-gram is hashed and MOSS selects a subset of these hash values as the program's fingerprints. Similarity is determined by the number of fingerprints shared by the programs, i.e., the more fingerprints they share, the more similar they are. For each pair of source-code fragments detected, the results summary includes the number of tokens matched, the number of lines matched, and the percentage of source-code overlap between the detected file pairs.

YAP3 converts programs into a string of tokens and compares them by using the token matching algorithm, Running-Karp-Rabin Greedy-String-Tiling algorithm (RKR-GST), in order to find similar source-code segments. YAP3 pre-processes the source-code files prior to converting them into tokens. Pre-processing involves removing comments, translating upper case letters to lower case, mapping synonyms to a common form (i.e., function is mapped to procedure), reordering the functions into their calling order, and removing all tokens that are not from the lexicon of the target language (i.e., removing all terms that are not language reserved terms). YAP3 was developed mainly to detect breaking of code functions into multiple functions, and to detect the reordering of independent source-code segments. The algorithm works by comparing two strings (the pattern and the text) which involves searching the text to find matching substrings of the pattern. Matches of substrings are called tiles. Each tile is a match which contains a substring from the pattern and a substring from the text. Once a match is found the status of the tokens within the tile are set to mark. Tiles whose length is below a minimum-match length threshold are ignored. The RKR-GST algorithm aims to find maximal matches of contiguous substring sequences that contain tokens that have not been covered by other substrings, and therefore to maximize the number of tokens covered by tiles.

# CHAPTER THREE
# PLAGIARISM ISSUE IN SOURCE CODES

With the development of technology, amount of software application have shown huge increase. However, it is seen that a plagiarisim subject begins to be big trouble in software area. We can see examples in generally big companies. While a company pay a lot of money to software otomation, they want that the application must belong only to them. However, other companies can take it and show themselves as owner of software. This situation cause a big problems and actions or proceedings are started. We can see other examples in education area. While students do their programming homework or project, they can take other one's works and show it as their study. This bothers instructors and they get worried while charging a studeny as cheater.

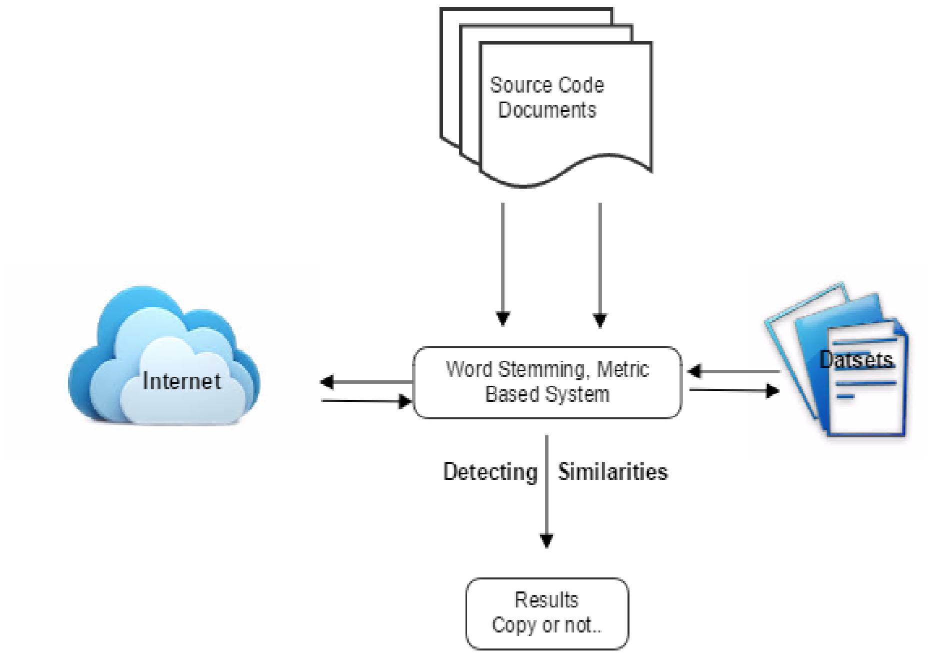


Figure 3.1 Work plan of copy code detection tool

To prevent to theft of source codes, new methods are allways produced. The base of this methods is finding similarity scores among source codes and analyzing the results. It is determined to existence of copy codes in dataset after comparing method’s outputs. Figure 3.1 shows that work plan of methods of copy code detection tools.

## 3.1 Definition of Plagiarism

Plagiarism is presenting someone else’s work or ideas as own, with or without their consent, by incorporating it into their work without full acknowledgement. All published and unpublished material, whether in manuscript, printed or electronic form, is covered under this definition. Plagiarism may be intentional or reckless, or unintentional. Under the regulations for examinations, intentional or reckless plagiarism is a disciplinary offence.

Plagiarism can take many forms, including the following;

- Word-by-word copying, which involves directly copying sentences or chunks of sentences from other peoples work without providing quotations and/or without appro priately acknowledging the original author.

- Paraphrasing, which involves closely rewriting (i.e. only changing some of the words but not making enough changes) text written by another author and appropriately citing the original author.

- Plagiarism of secondary sources, which involves referencing or quoting original sources of text taken from a secondary source without obtaining and looking up the original source.

- Plagiarism of the form of a source, is when the structure of an argument in a source is copied without providing acknowledgments that the ‘systematic dependence on the citations’ was taken from a secondary source. This involves looking up references and following the same structure of the secondary source.

- Plagiarism of ideas, which involves using ideas originally expressed in a source text without ‘any dependence on the words or form of the source’.
- Blunt plagiarism or authorship plagiarism which is taking someone else’s work and putting another’s name to it.

## 3.2 Plagiarism in Software Area

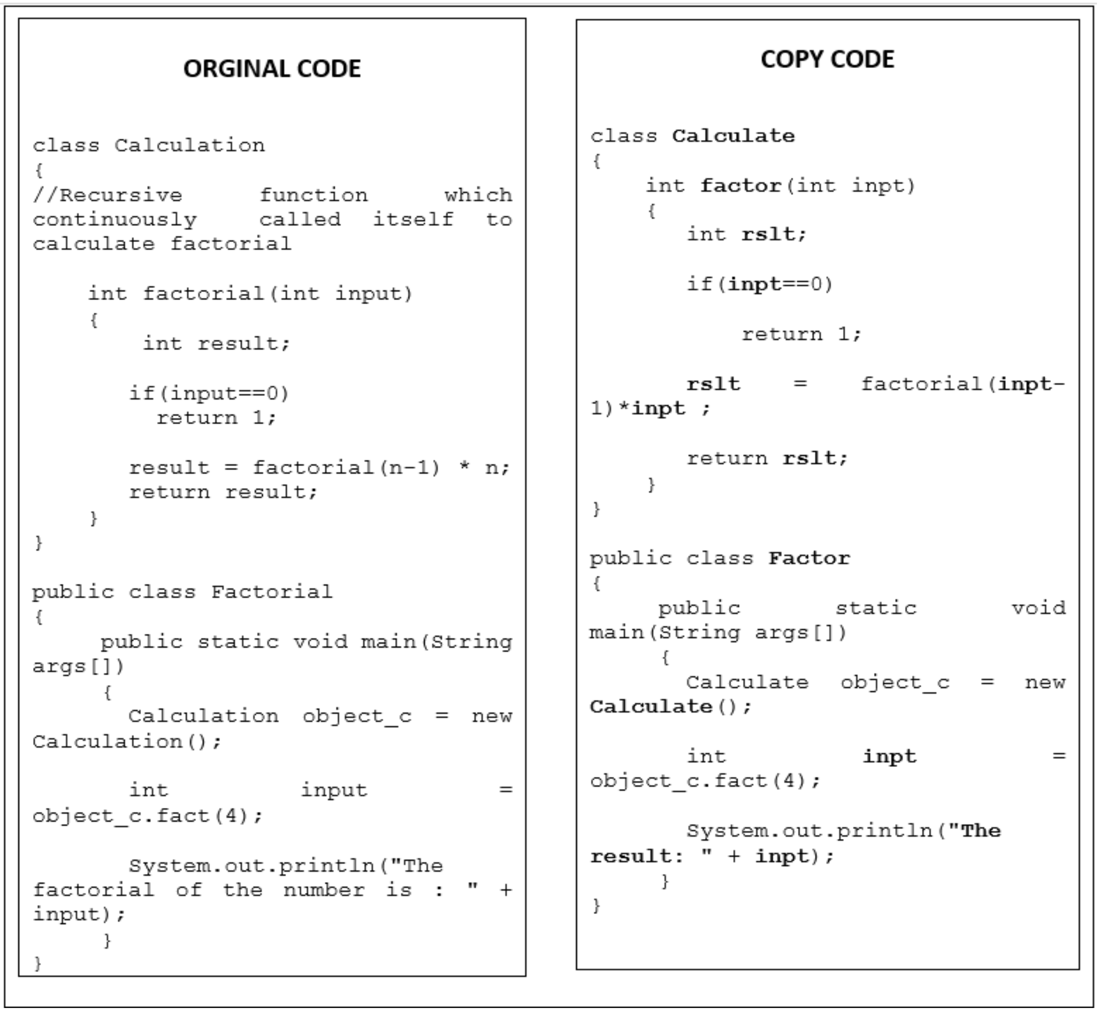
Plagiarisim is a crime that is at issue not only in academic area but also in literature, art and every field of industry. One of the plagiarism types is source code theft. Source code theft is producing a new program by modifiying a software application without permission of code owner. Development of an application needs a lot of works and stealing all or a part of a program is unwanted situation by programmers.

When it is compared with other text types, it can be said that a computer program has more rule based. So, it must be more carefull while evaluating source code documents to detect plagiarism. There are some tools are available to prevent code thefts nowadays. These tools are concentrate on obtaining similarity among programs or guessing who coder is. However, there is not available certain solution againist source code cloning yet.

## 3.3 Plagiarism in Education Area

Plagiarized code is a modified and concerted version of original code which is taken without permission of code owner.  Especially, in education area, some students, who attend programming courses, copy all or part of a program from other students and submit the copy as their own work. When the copy code is inspected in detail, it can be obviously seen that most of students only change the specific points of program such as renaming variable names, adding comment lines, replacing code blocks etc.., while creating copy codes.

At this stage of the study, it is performed a work with a team consisting of five instructors. It is determined which parts of source code are generally modified by the students while creating copy code. Instructors highlight that they realize the most noticeable thing is renaming variables, functions and parameters when they inspect copy codes. In this part, renaming is the most efficient step while generating dataset that includes modified code examples to test study. For example, in Figure 1, both of the program codes calculate the factorial of a number.  While left one is original code, the code in the right side is copy. At first glance, distinction in identifiers can be realized easily.  It is clearly seen that the name of parameter “input” is changed as “inpt” and parameter “result” is changed as “rslt” while producing a new code from the original one.  Also, when looking at similarities among source codes, regulating comment lines like removing, translating into another language etc… is efficient for decreasing similarity ratio. In Figure 3.2, it is also seen that the comment line which gives information about “factorial” method is removed at the right code.



**ORGINAL CODE**

```
class Calculation
{
//Recursive     function     which
continuously    called  itself  to
calculate factorial

    int factorial(int input)
    {
        int result;

       if(input==0)
         return 1;

       result = factorial(n-1) * n;
       return result;
    }
}

public class Factorial
{
     public static void main(String
args[])
     {
       Calculation  object_c  =  new
Calculation();

       int         input           =
object_c.fact(4);

       System.out.println("The
factorial  of  the  number  is  :  "  +
input);
     }
}
```

**COPY CODE**

```
class Calculate
{
    int factor(int inpt)
    {
       int rslt;

       if(inpt==0)

           return 1;

       rslt     =     factorial(inpt-
1)*inpt ;

       return rslt;
    }
}

public class Factor
{
     public       static       void
main(String args[])
      {
       Calculate   object_c   =   new
Calculate();

       int          inpt           =
object_c.fact(4);

       System.out.println("The
result: " + inpt);
      }
}
```

Figure 3.2 Comparing original and copy code

As a result of inspections, the following steps are selected which are common methods to generate copy source codes. These steps are;

- Renaming identifiers,
- Adding or removing blank lines,
- Modifying the comment lines,
- Changing parameter order in functions/methods,
- Removal of functions/methods,
- Adding or removing operator space.

## 3.4 Relation between Source Code Similarity and Source Code Plagiarism

Day by day, software plagiarism issue becomes apperant in many fields such as universities, private corporation, corporate firms etc… To understand whether a code is copy, the similarity ratios between current code and other ones are counted. Acording to similarity ratios among the source codes, it can be understood if the program is plagiarized or not.

There is important point that similarities among source codes don't always reflect plagiarism issue. A developer can take a part of an open source code while coding any project. Other example that if a programmer let to take a part or all of his code, there is no problem to use it by the others. However, this subject can differ in education area. Generally, instructors don't prefer that students use other sources completely in their homeworks, projects etc… Because they want to be fair while evaluating the submissions. Of course, this approach is completely true in education area because the homeworks is given to teach something not copy. Also, utilising web while preapering a homework is inevitable. In fact, there is no affair in that point. However, more than one students can use the same web site and can take the same example while preapering their submission. When instructor check the assignments to control if they are copy or not, s/he will see that the homeworks that is obtained by same web sites is so similar and s/he will score them as copy.

Briefly, the similarity and plagiarism issues can get involved. However they are not same subject. In education area, restrictions are used for that the students do their homework with their efforts and don’t show other’s codes as their own. So, in this area similarity and copy terms can be consideres synonyms inverse of other areas.

## 3.5 Conclusion

Eventually, plagiarism issue is so impartant nearly every parts of our life. In literature area, an author doesn’t want his work to be in use by the other ones.  A scientist absolutely oppose if avent garde idea is mentioned in his articles is used by the other persons without his permission of him or reference. A programmer who developed special software, spends a lot of effort on merchantable programs. Of course s/he doesn’t allow to be received his codes by the other developers. In this study, the critical point is understanding if software product is copy or not. If people or corporation let to use of their informations and inputs by the other, we can not identify it as plagiarism. However, taking some parts or entire of source code is not permitted, the plagiarism issue is revealed.

# CHAPTER FOUR
# TEXT MINING

## 4.1 Definition of Text Mining

To deduce valuable information from a text document, it is necessary to apply some processes before. While applying these processes, methods belong to Text Mining are used. Text mining or knowledge discovery from text (KDT) is mentioned by Feldman et al. for the first time. It uses techniques from information retrieval, information extraction as well as natural language processing (NLP) and connects them with the algorithms and methods of KDD, data mining, machine learning and statistics.

Generally, Text Mining works on documents that written by using natural language.The methods of KDD clean the documents and eliminate unnecessary, garbage datas from the text. In addition to cleaning process, it turns the unstructured document to structured documents to use data mining techniques on it. Figure 4.1 shows the steps of Text Mining.

Text Mining can be inspected on two main classes:

- Understanding/Summarizing Text: One aim of text mining is extracting valuable information. So, the key which is included by text can be understood easily.

- Modelling with Text: Generally, text mining can be used to determine. A system of shopping web site that predict customers' behaviours can be developed by using Text Mining methods. Contents of text are used as input variable and the projected model is extracted by these informations.

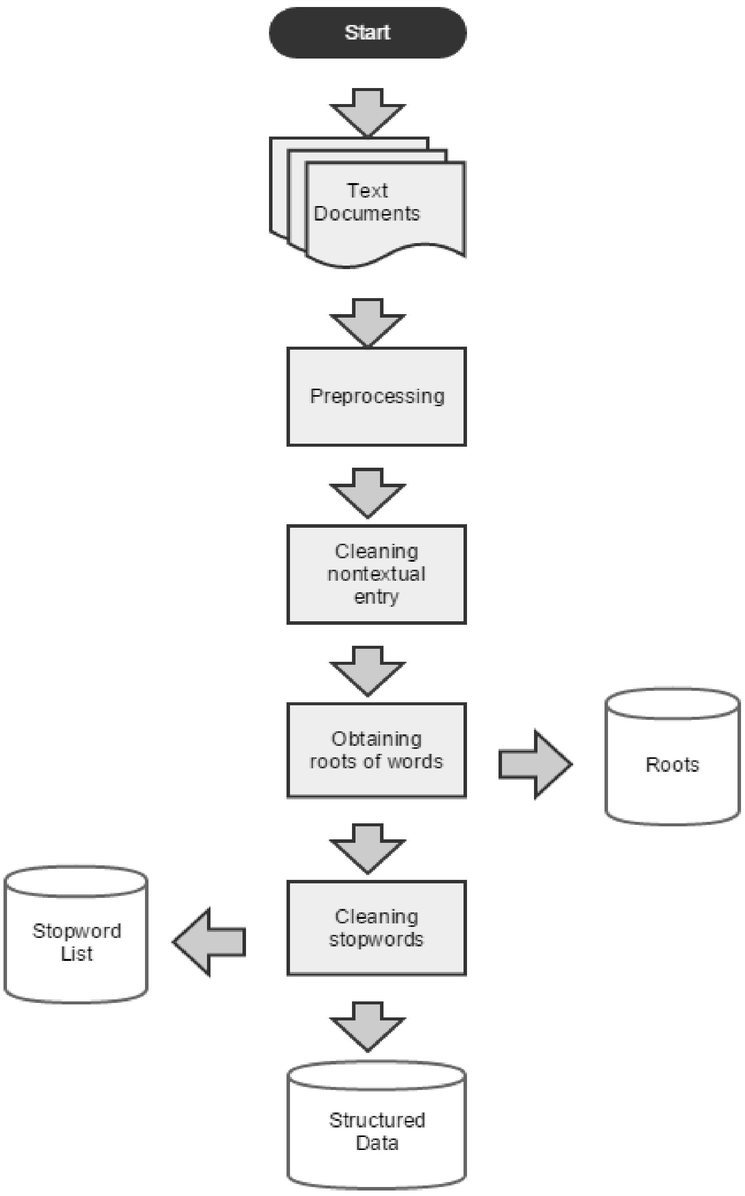


Figure 4.1 Flow diagram of text mining

## 4.2 Application Area of Text Mining

Text Mining generally presents sollutions for needs of scientific researchs and businesss world. It is used in many areas such as below.

Customer Relationship Management: It extracts information from emails, web surfings, survey of all customers. These qualified informations can be used in fathoming out customers' need.

Detecting Frauds: Some personel informations are stored in big datawarehouses such as health institutions and organizations, banks etc… These informations can be stolen and used by the ill-wisher persons. However, with text mining methods it can realize when it is anormal situations in system. So the problems can be overcome.

Scientific Research: When a subject want to be researched, necessary information can be extracted from topic or contents of articles and publications.

Security: Text mining methods can be used for predicting cyber attacks, criminal behaviours, terrorist attacks etc… If this situations can be understood early thanks to text mining methods, it can be prevented easily.

## 4.3 Methods of Text Mining

### *4.3.1 Natural Language Processing*

Natural Language Processing is a sub-branch of computer and language sciences. Natural Language Processing is a theoretically motivated range of computational techniques for analyzing and representing naturally occurring texts at one or more levels of linguistic analysis for the purpose of achieving human-like language processing for a range of tasks or applications.

Some of work areas of NLP are:

- Paraphrase an input text,
- Translate the text into another language,
- Answer questions about the contents of the text,
- Draw inferences from the text.

NLP works firstly taking texts as input. And parsing processes are started. After parsing process the obtained parts of texts are evaluated and made sense of. Figure 4.2 shows the steps of NLP briefly.

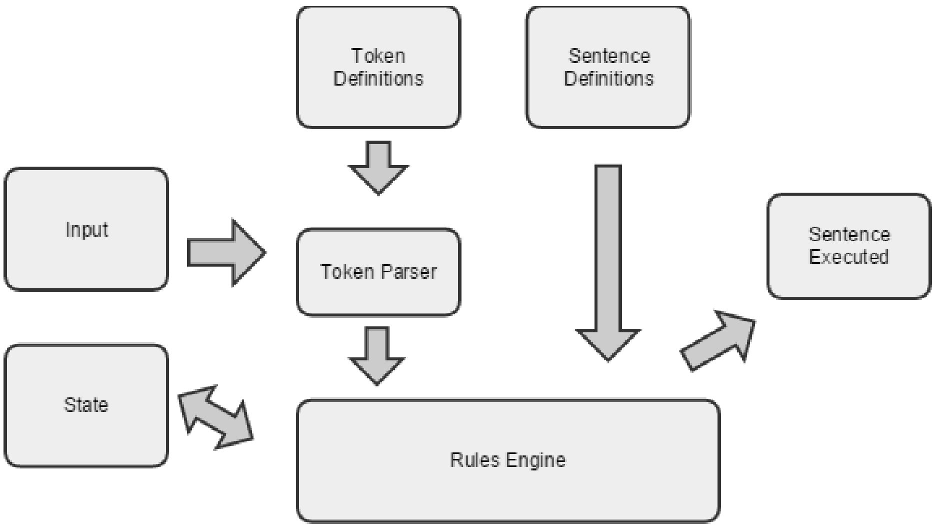


Figure 4.2 Steps of NLP

*4.3.1.1 N- gram Algorithm*

N-gram algorithm obtains a substring combination and finds repeat ratios of this substring in a character array which will be compared with other strings to find similarity. Besides using in fields of natural language processing, owing to technological development, N-gram algorithms have started to be used in programming languages. The algorithm inspects documents to categorize and to find similarities. N-gram algorithm is recognized as one of the simplest and best efficient method that finds similarity among strings.

An N-gram algorithm starts to work with dividing a text into substrings has length of N that is specified by the user. When reached to N-1th element of string, process is terminated. If value of N is one, it is called uni-gram. If value of N is two, it called bi-gram. If value of N is three, it is called tri-gram. For example, to explain tri-gram, the results in Table 4.1 can be shown.

Table 4.1 Tri-grams in STRING word and frequencies of substrings

| N-Gram | Frequencies |
|---|---|
| **STR** | 1 |
| **TRI** | 1 |
| **RIN** | 1 |
| **ING** | 1 |

As shown in Table 4.1, at fourth character of STRING, grouping process is terminated so, N-gram algorithm is completed. Substrings in specified text and frequencies of them are held to compare two or more documents.

At indexing step, the documents are partitioned into N-grams, and then each N-grams word is added to lists correspondingly. Figure 4.3 explains the indexing step briefly. At search step, the query is also partitioned into N-grams, and for each of them corresponding lists are scanned using the metric.

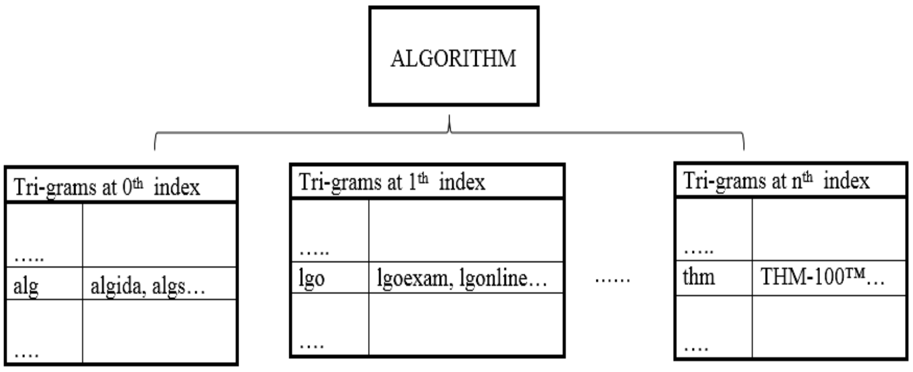


Figure 4.3 Tri-grams in "ALGORITHM"

Briefly, in this algorithm, N-gram frequencies of two documents are compared and distances between them are measured. The distance variable takes value between 0 and 1. While the value of distance closes to 1, it is deduced that similarity ratio increases. Otherwise, this ratio decreases.

In this study, the reason of choosing N-gram algorithm is providing language independent structure and obtaining accurate results while finding similarities among source code documents. Value of N is specified as three and tri-grams in each source codes are compared separately to obtain similarity ratios.

### *4.3.2 Information Retrieval*

Information retrieval (IR) is the activity of obtaining information resources relevant to an information need from a collection of information resources. Searches can be based on metadata or on full-text (or other content-based) indexing.

An information retrieval process begins when a user enters a query into the system. Queries are formal statements of information needs, for example search strings in web search engines. In information retrieval a query does not uniquely identify a single object in the collection. Instead, several objects may match the query, perhaps with different degrees of relevancy.

An object is an entity that is represented by information in a database. User queries are matched against the database information. Often the documents themselves are not kept or stored directly in the IR system, but are instead represented in the system by document surrogates or metadata.

Most IR systems compute a numeric score on how well each object in the database matches the query, and rank the objects according to this value. The top ranking objects are then shown to the user.

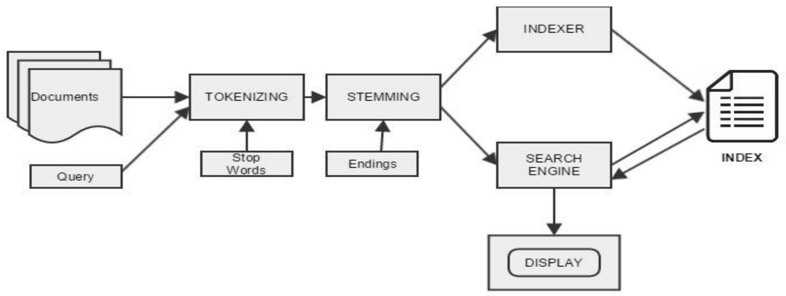


Figure 4.4 Information retrieval system

### *4.3.3 Information Extraction*

Extract method is discovering relevant information and ignoring non-relevant information in a textual data. The inputs in this method are well defined text based queries. At the end of this process link related information and output in a predetermined format are obtained.

Figure 4.5 shows steps of Information Extraction Model. The model takes documents and combined it query results. So, ranked documents are created. Shortly, all processes in this model are created to complete extraction of partial knowledge in the text.

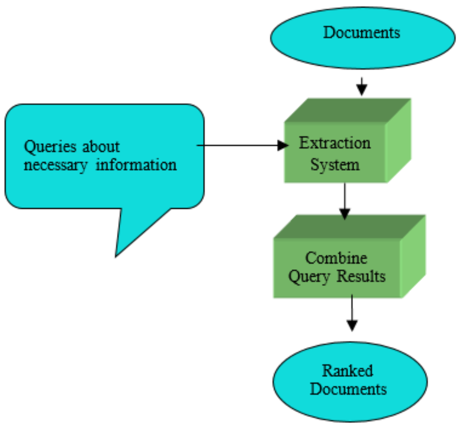


Figure 4.5 Information extraction method

### *4.3.4 Web Mining*

Development of technology causes enormous textual information on the Web. For example, when we look for an information from the Web, we can find a lot of sources such as books, videos, articles, blogger comments etc…

Briefly web mining can be defined as information retrieval of textual documents and extraction of partial knowledge using the web.

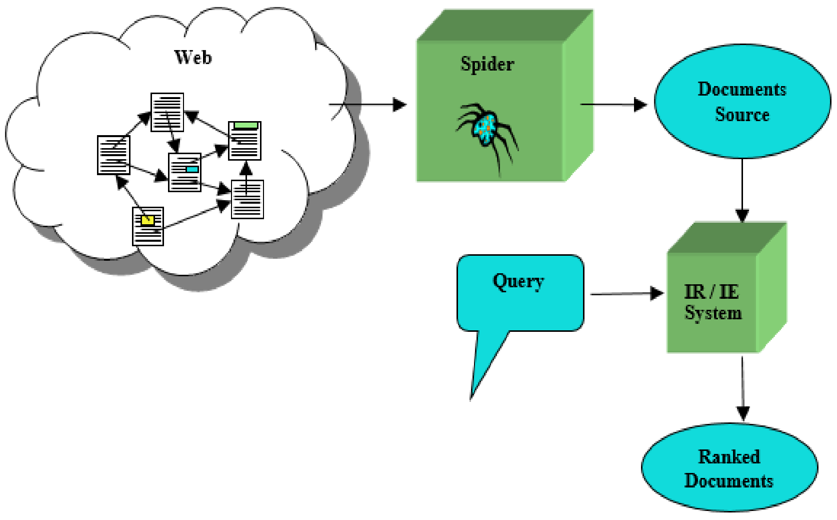


Figure 4.6 Web mining method

## 4.4 Using of Vector Space Model in Text Mining

The representation of a set of documents as vectors in a common vector space is known as the VSM and is fundamental to a host of information retrieval operations ranging from scoring documents on a query, document classification and document clustering. In this model, each dimension shows a separate term. All terms in vector have a weight that is represented as "*w*". According to query result, if a document contains a term, value of weight is counted and takes a value different from 0. In Figure 4.7, x and y show two documents that will be compared and $w_1$, $w_2$ and $w_3$ indicate the weight of terms in documents.

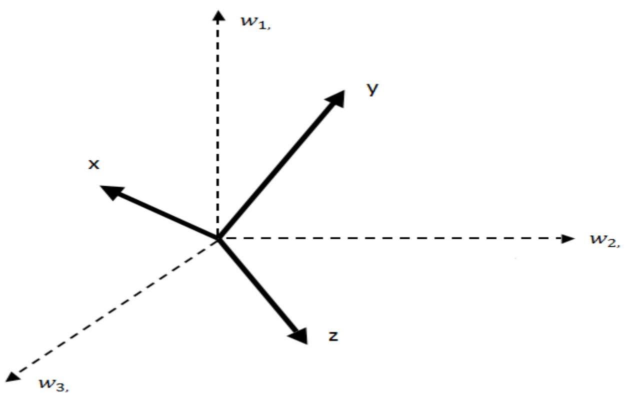


Figure 4.7 Vector space model

Weights of terms can be calculated by TF×IDF method. Term Frequency (TF) represents frequency of a term in the document. Inverse Document Frequency (IDF) gives information about the number of times that term occurs in all documents of collection. Equation of (TF) and (IDF) are shown in Equation 4.1 and Equation 4.2.

$$TF(t) = \frac{Number\ of\ times\ term\ t\ appears\ in\ a\ document}{Total\ number\ of\ terms\ in\ the\ document} \qquad (4.1)$$

$$IDF(t) = \log_e \frac{Total\ Number\ of\ documents}{Number\ of\ documents\ with\ term\ t\ in\ it} \qquad (4.2)$$

$$\begin{array}{c|cccc} & \mathrm{T}_1 & \mathrm{T}_2 & \cdots & \mathrm{T}_3 \\ \mathrm{D}_1 & \mathrm{d}_{11} & \mathrm{d}_{12} & \cdots & \mathrm{d}_{1t} \\ \mathrm{D}_2 & \mathrm{d}_{21} & \mathrm{d}_{22} & \cdots & \mathrm{d}_{2t} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \mathrm{D}_n & \mathrm{d}_{n1} & \mathrm{d}_{n2} & & \mathrm{d}_{nt} \end{array}$$

Figure 4.8 Document term matrices

Document term matrix is seen in Figure 4.8. In this matrix, T represents terms in documents, D shows documents in datasets and d indicates the frequencies of terms in documents.

After finding the weights of all terms in vector, some vector operations are used to compare documents to specify how they are similar. Generally, cosine of the angle between documents is calculated. This method is called Cosine Normalization. Equation 4.3 shows formula of Cosine Normalization.

$$sim_{d1,d2} = \cos Q = \frac{d1\ d2}{IId1II\ IId2II} \qquad (4.3)$$

# CHAPTR FIVE
# EXTRACTING METRIC METHODS

## 5.1 Definition of Software Metrics

Extracting software information about previous conditions of software and using obtained outputs for predicting software's future are so important in software development life cycle. So, there are many knowledge about a program and analyzing process can be done easily. Correspondingly, having detailed datas about software life cycle provides convenience to project managers in decision stage. Software metrics represent units about developed software to scale and control quality of product.

Code line count, value of complexity, error rate, cohesion are example of software metrics. These metrics are significant to have information about programs. Software projects can be kept down thanks to software metrics.

## 5.2 Using Software Metrics to Find Similarity between Source Codes

Software metrics are not only used in steps of software development life cycle. They are used in software plagiarism issues, because they give clues about source codes. Characteristics of programmer can be obtained by using metrics.

Source code author identification is much harder than natural language authorship attribution or writer identification (of handwriting) or even speaker recognition. The traditional methodology that has been followed in this area of research is divided into two main steps .The first step is the extraction of software metrics representing the author's style and the second step is using these metrics to develop models that are capable of discriminating between several authors, using a classification algorithm.

There are certain patterns that developers inherently produce based on their particular style of coding while still following the guidelines, rules and grammar of the language. Similar to analyzing prose for authorship it can be identified certain peculiarities in the styles of software developers and use these styles determine the authorship found in the source code of developers.

General metric examples are;

- Total Lines Count
- Code Lines Count
- Comment Lines Count
- Whitespace Lines Count
- Code / Comment Ratio
- Code / Total Ratio
- Function Code Lines Count
- Function Return Types Number (void, int, double…)
- Style of Variables(Camel or Pascal)
- Curly Bracket Count
- If-Else Count
- Global Variables
- Local Variables
- Global/Local Variables Ratio
- Average Indentation
- For Count
- While Count

## 5.3 Metrics Types

Metrics can be inspected in two categories. Some metrics give information about general view of source code such as indentation style, placing comment lines.

### *5.3.1 Settlement Metrics*

Programming Layout Metrics gives information general view of source codes in code editor. When it is looked at a source code for the first time, style of code owner can be understood. For example when the code in Figure 5.1 is inspected, it can be realized at first glance that coder puts blanks lines after all code lines.  Also when the looked at code in Figure 5.2, it can be easily said that programmer chooses to write comments below the code lines. That means the owner of code in Figure 5.2 places importance on commenting style. These types of metrics reflect the source code view on the editor and is named as settlement metrics.

```
int strchk(char *s1)
{
char *ptr1;

ptr1 = s1;

while(*ptr1 != 0)

if(*ptr1++ == '\t')

return(1);

return(0);
}
```

Figure 5.1 Added blank lines code

```
/* Checks the existence of \t in string */
int check_for_tab_in_string(string)
char *string;
{
char *character_pointer;
/* Loop until found or we reach EOLN */
for(character_pointer = string;*character_pointer != NULL;)
{
/* check to see if we found TAB */
if(*(character_pointer++) == 9)
{
/* Success!! */
return(TRUE);
}
```

Figure 5.2 Comment writing style

In Table 5.1, Table 5.2 and Table 5.3, other examples of metrics and their definitions are come up.

Table 5.1 Settlement metrics

| Settlement Metric | Definition |
|---|---|
| Metric ST1 | It indicates the indentation style. It consists of 9 sub-metrics |
| Metric ST2 | It shows if Else keyword is written as indented |
| Metric ST3 | It shows that the variables written as indented |
| Metric ST6 | It gives information about comment line indentation style |
| Metric ST7 | It gives information about blank line counts |

Table 5.2 Metric ST1 sub-branches

| Metric ST1 Sub-branch | Definition |
|---|---|
| Metric ST1a | It shows indentation of block code lines. |
| Metric ST1b | It counts number of open curly brackets that is used alone in a code line. |
| Metric ST1c | It shows percentage of open curly bracket that is used alone in a code row. |
| Metric ST1d | It shows percentage of open curly bracket that is last element in a code row. |
| Metric ST1e | It counts number of closed curly brackets that is used alone in a line. |
| Metric ST1f | It shows percentage of closed curly bracket that is used alone in a code row. |
| Metric ST1g | It shows percentage of open closed bracket that is last element in a code row. |
| Metric ST1h | It gives information about open curly brackets indentation style. |
| Metric ST1i | It gives information about closed curly brackets indentation style. |

Table 5.3 Metric ST6 sub-branches

| Metric ST6 Sub-branch | Definition |
|---|---|
| Metric STY6a | Information about using frame while creating comment lines. |
| Metric STY6b | It stores percentage of aligned comment lines in program. |
| Metric STY6c | Block style comment lines to all code lines ratio. |

### *5.3.2 Programming Style Metrics*

This type of metrics dwell on context program codes. In this part of study, it is dealed with points such as which type of naming style (camel, pascal, etc...), loop (for, while, do-while, etc…) the programmer generally prefers while coding, what is the count of average length of identifier names in code etc… This kind of informations reveal the user’s characteristics while writing program codes. So, depending on programmer style metrics, it can be easily found out who code owner is. In this part, striking feature is information about naming styles. Table 5.4, Table 5.5 and Table 5.6 shows other Programming Style Metrics examples.

Table 5.4 Programming style metrics

| **Programming Style Metric** | **Tanım** |
|---|---|
| Metric PRO1 | Average program lines count. |
| Metric PRO2 | It gives information about names of variables. It can be inspected in 4 sub-metrics. |
| Metric PRO3 | It gives information about naming style of programmer. It can be inspected in 4 sub-metrics. |
| Metric PRO4 | Global variable count to local variable count ratio. |
| Metric PRO5 | Global variable count to code line number ratio. |
| Metric PRO7 | Aynı amaca hizmet eden while, for ve do döngülerinin kullanılış önceleği bilgisini içeren metriktir. |

Table 5.5 Metric PRO2 sub-branches

| **Metric PRO2 Sub-branch** | **Definition** |
|---|---|
| Metric PRO2a | It gives information about average length of local variables. |
| Metric PRO2b | It gives information about average length of global variables. |
| Metric PRO2c | It gives information about average length of function names. |
| Metric PRO2d | It gives information about average length of function parameters’ names. |

Table 5.6 Metric PRO3 sub-branches

| **Metric PRO3 Sub-branch** | **Definition** |
|---|---|
| Metric PRO3a | It indicates whether the underscore is used in naming. |
| Metric PRO3b | It indicates whether the coder uses a synonyms of identifier names. (for example tepm->tmp) |
| Metric PRO3c | Percentage of count of variable names that are started with upper case. |
| Metric PRO3d | Percentage of count of function names that are started with upper case. |

# CHAPTER SIX
# EXPERIMENTAL STUDY

## 6.1 Experimental Datasets

In this study, there are two datasets for experimental study.

First one is existing from students' homework projects belong to Algorithm and Programming Course.

Second one is modified dataset that is generated from a code example by changing specied points in the code.

### *6.1.1 Dataset Belong the Students Homework*

In this study, first of all N-gram algorithm is tried on two different datasets. Datasets consist of programming assignments of students who attend Software Engineering Department in Celal Bayar University.

In this part, first dataset includes five source code documents. These documents are obtained by changing a code example that belong to student who takes Algorithm and Programming Course. Five copy codes are derived from an original one. While preapering the dataset contains five document, the aim is labeling the all code documents from non-copy to copy. For this purpose, program is changed in in various forms. Labels are respectively copy, very similar, similar, little similar, not copy.

Second dataset contain all programming homeworks of freshmen and upperclassmen. While freshmen do their assignments by using C programming language, upperclassmen use C#.

Owing to N-gram algorithm provide flexibility for all programming language, all document in dataset of this part can be easily compared.

### *6.1.2 Modified Dataset*

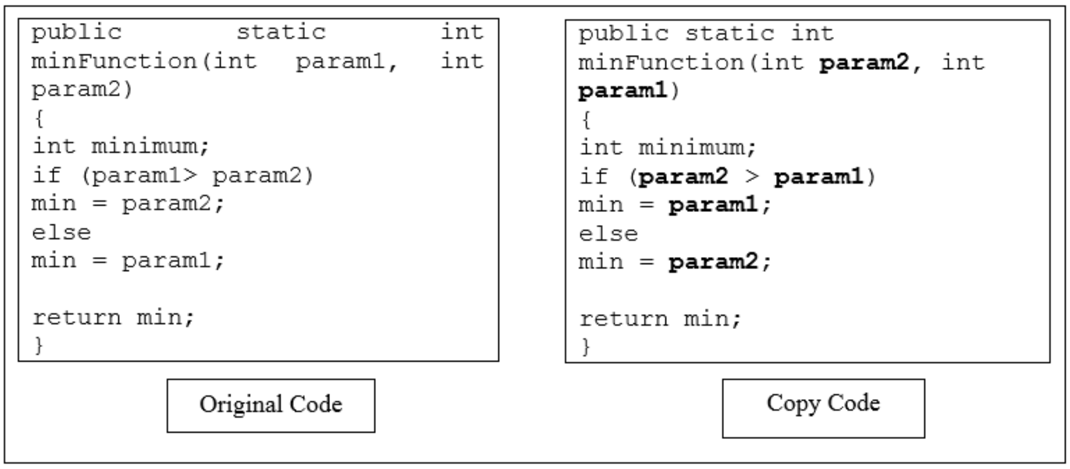
In this part of the study, 63 different code examples are generated by modifying the original code example as reported in the third section of paper. The documents which are created to test the study and to view similarity scores are called according to their alteration style. Table 6.1 shows all acronyms of documents names and explanations.

Table 6.1 Document name synonyms and explanations

| Document Name | Explanation |
|---|---|
| $D_{BL}$ | Adding/Removing Blank Line |
| $D_{RI}$ | Renaming Identifiers |
| $D_{PO}$ | Changing Parameter Order |
| $D_{OS}$ | Adding/Removing Operator Space |
| $D_{RF}$ | Relocation of Functions |
| $D_{MC}$ | Modifying Comment Lines |
| $D_{C1}$ | Adding/Removing Blank Line + Renaming Identifiers |
| $D_{C2}$ | Adding/Removing Blank Line + Changing Parameter Order |
| $D_{C3}$ | Adding/Removing Blank Line + Adding/Removing Operator Space |
| $D_{C4}$ | Adding/Removing Blank Line + Relocation of Functions |
| $D_{C5}$ | Adding/Removing Blank Line + Modifying Comment Lines |
| $D_{C6}$ | Adding/Removing Blank Line + Renaming Identifiers + Changing Parameter Order |
| $D_{C7}$ | Adding/Removing Blank Line Renaming Identifiers + Adding/Removing Operator Space |
| $D_{C8}$ | Adding/Removing Blank Line + Renaming Identifiers + Relocation of Functions |
| $D_{C9}$ | Adding/Removing Blank Line + Renaming Identifiers + Modifying Comment Lines |
| … | ……………………………………………………………………….. |
| … | ……………………………………………………………………….. |
| $D_{C57}$ | Adding/Removing Blank Line + Renaming + Parameter Order + Operator Space + Functions + Comment Lines |

In Table 6.1, first six documents are obtained by adding or removing blank line, changing parameter orders in methods, adding or removing spaces between operators, relocation of functions and modifying comments. Other 57 ones are generated by combining of the six alteration steps. According to the Table 6.1, $D_{BL}$ represents the documents which are formed with additional blank lines among original source code lines.

$\boldsymbol{D_{RI}}$ demonstrates the code obtained from original source code by changing identifiers' names. $\boldsymbol{D_{PO}}$ shows the a code document yielded by changing the location of the parameters of methods in original code. For example, let's look at the Figure 6.1 includes a method that finds the minimum between two numbers. While "minFunction" method in original code includes parameters "param1" and "param2" sequentially, the copy code this parameter order relocated.



```
public        static        int
minFunction(int   param1,   int
param2)
{
int minimum;
if (param1> param2)
min = param2;
else
min = param1;

return min;
}
```

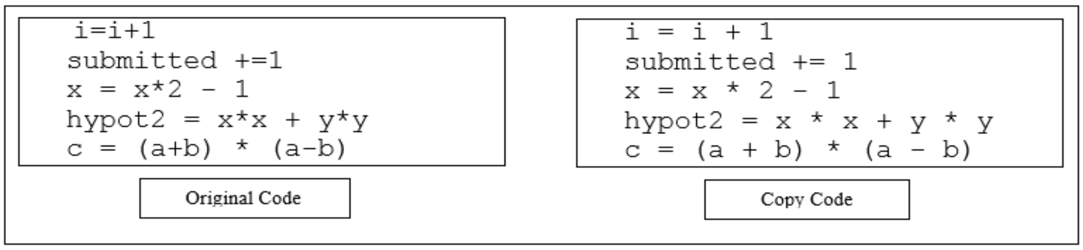
Original Code

```
public static int
minFunction(int param2, int
param1)
{
int minimum;
if (param2 > param1)
min = param1;
else
min = param2;

return min;
}
```

Copy Code

Figure 6.1 Comparing original and copy code (1)

$\boldsymbol{D_{OS}}$ implies spacing out before or after operators. For example, it can be easily realized in Figure 6.2. While code block at the left side of Figure don't include any space before or after operator, at the left it can be easily recognized that there are spaces between operators.



```
 i=i+1
submitted +=1
x = x*2 - 1
hypot2 = x*x + y*y
c = (a+b) * (a-b)
```

Original Code

```
i = i + 1
submitted += 1
x = x * 2 - 1
hypot2 = x * x + y * y
c = (a + b) * (a - b)
```

Copy Code

Figure 6.2 Comparing original and copy code (2)

$\boldsymbol{D_{RF}}$ indicates changing of place of functions or methods in code lines. For example if a function starts at 57th code lines in original code, the student who attempt to plagiarize can move it 3rd code lines to show code as different. In the opinion of instructors, this is one of the most common methods among students while attempting to copy a code. $\boldsymbol{D_{MC}}$ is obtained by modifying comment lines as contraction or rewriting in different language.

## 6.2 Experimental Results

### *6.2.1 Results of Dataset Belong the Students Assignments*

While inspecting first dataset that is created by modifying a specified code, the results are formed by Fuzzy Logic Approach as seen in Table 6.2. In deriving new codes from original code, process is started with fundamental change and stopped with little change in original code. Five categories are formed with changing original code.

These categories are;

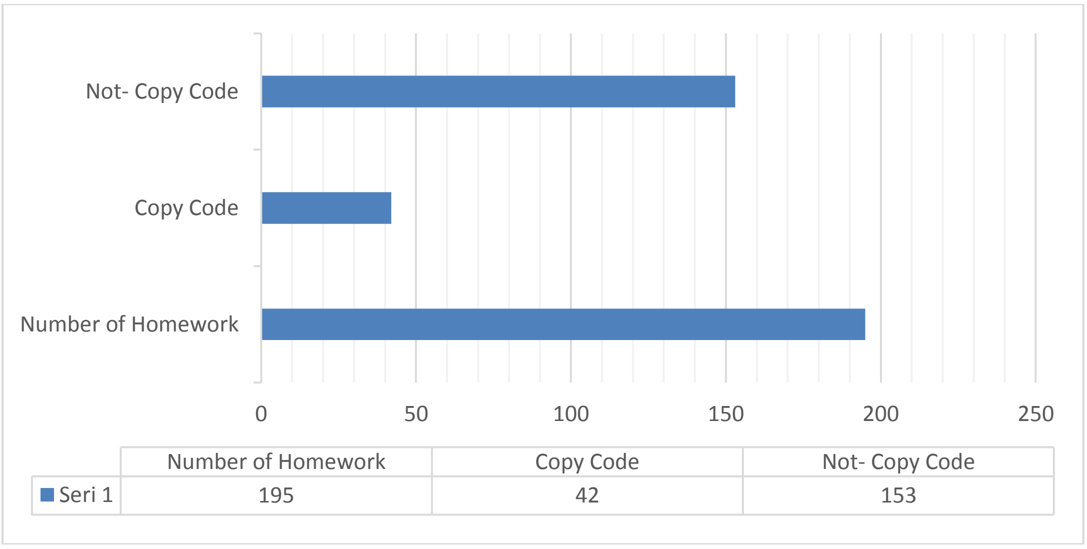
- Copy,
- Very similar,
- Similar,
- Little Similar,
- Not copy.

N-gram results of codes in each categories and the real values are compared to test the accuracy of study. With consequence assessments, it is seen that real sequence from copy to not copy in categories is the same with sequence formed with N-gram algorithm.

Table 6.2 Compare results of first dataset

| Document | Original Document |
|---|---|
| Code 1 (Copy Code) | **100%** |
| Code 2 (Very Similar) | **91.2%** |
| Code 3 (Similar) | **84.8%** |
| Code 4 (Little Similar) | **69.4%** |
| Code 5 (Not Copy) | **43.7%** |

While evaluating second dataset which contains homeworks of freshmen and upperclassmen in this part of study, expected result is seeing effects of language independence. Owing to N-gram algorithm expected results are obtained and programming assignments which are written by two different language are grouped correctly. With these results, it is realized that 42 of 195 assignments were copy. These results can be seen in Figure 6.3 and Figure 6.4. To validate these sitiuations, instructors of related courses discussed with students who cheater are according to result of program. After discussions, most of students accepted that they took the assignment from their friends and showed as their work. This results shows the success of N-gram algorithm detecting similarities and it can be worked on any documents, no matter which programming language is used.



|  | Number of Homework | Copy Code | Not- Copy Code |
|---|---|---|---|
| ■ Seri 1 | 195 | 42 | 153 |

Figure 6.3 Experimental result of dataset that contains of student’s homework

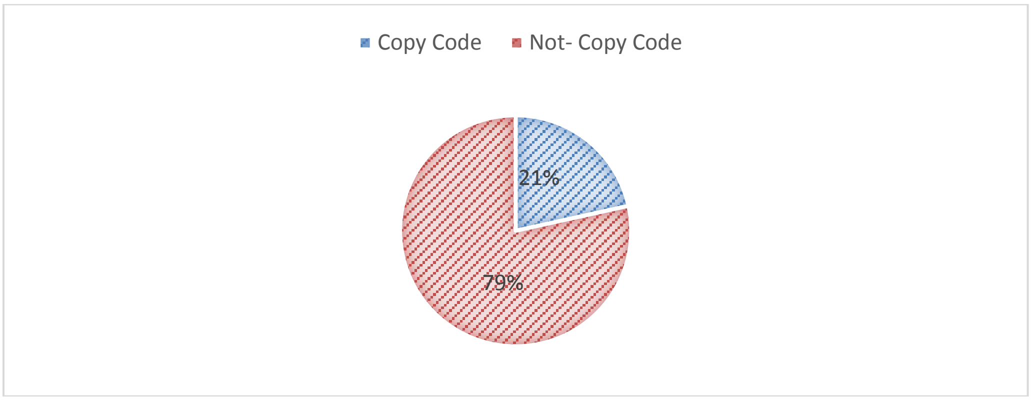


Figure 6.4 Percentage of copy code assignment

### *6.2.2 Experimental Results of Modified Dataset*

After creating dataset as mentioned in the previous section, Tri-gram and VSM Tri-gram similarity ratios are obtained and the results are showed in Table 6.3. The results of similarity ratios are between 0 and 1. When this value approximate to 1 from 0, it can be understood that the similarity is higher between two compared documents.

After the codes in datasets are inspected by the instructors it is obviously seen that VSM results are more consistent. For example, according to the instructors, adding space before or after operators is not effective while attempting to change source code. However, when it is looked at the Table 6.3, it can be realized that the similarity ratio between $\boldsymbol{D_{OS}}$ and original document is low significantly beside VSM Tri-gram result. Even the ratio of $\boldsymbol{D_{OS}}$ is smaller than $\boldsymbol{D_{C1}}$ which consist of adding/removing blank line and renaming identifiers. The other point the instructors especially indicate that modifying comment is more effective than adding or removing blank lines among the code lines. However in the tri-gram results, $\boldsymbol{D_{MC}}$ and $\boldsymbol{D_{BL}}$ has nearly same similarity score when they compare to original code and it is obviously seen that the difference in similarity values of $\boldsymbol{D_{BL}}$ and $\boldsymbol{D_{MC}}$ are more coherent in VSM Tri-Gram.

Table 6.3 Tri-Gram and VSM Tri-Gram results

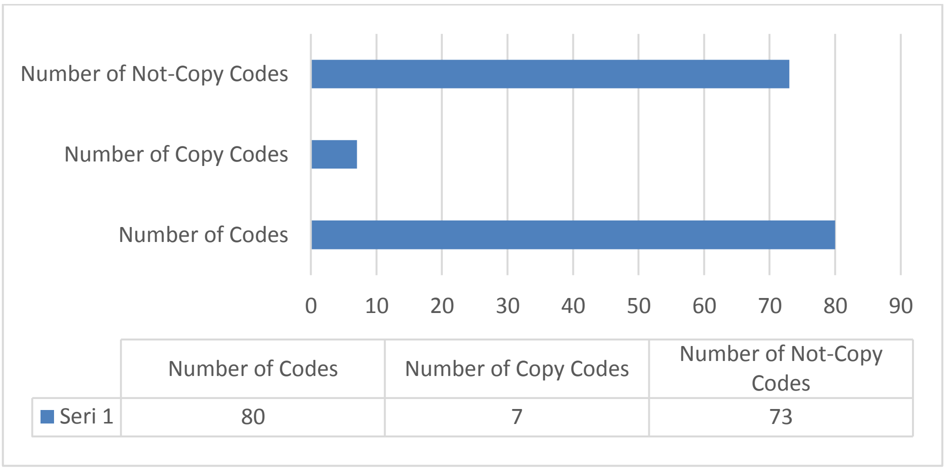
| Document(Original) | Tri-Gram Result | VSM Tri-Gram Result |
|---|---|---|
| $D_O$ | 1 | 1 |
| $D_{BL}$ | 0.96 | 0.98 |
| $D_{RI}$ | 0.86 | 0.88 |
| $D_{CP}$ | 0.94 | 0.97 |
| $D_{OS}$ | 0.81 | 0.96 |
| $D_{RF}$ | 0.93 | 0.94 |
| $D_{MC}$ | 0.95 | 0.92 |
| $D_{C1}$ | 0.82 | 0.90 |
| $D_{C2}$ | 0.94 | 0.93 |
| $D_{C3}$ | 0.80 | 0.94 |
| $D_{C4}$ | 0.90 | 0.92 |
| $D_{C5}$ | 0.94 | 0.91 |
| $D_{C6}$ | 0.76 | 0.84 |
| $D_{C7}$ | 0.72 | 0.83 |
| $D_{C8}$ | 0.74 | 0.81 |
| $D_{C9}$ | 0.78 | 0.80 |
| $D_{C10}$ | 0.69 | 0.77 |
| ... | ... | ... |
| ... | ... | ... |
| $D_{C6}$ | 0.69 | 0.66 |

As mentioned in Section 3.3, instructors claim that students generally choose only one alteration type while trying to change source code. Commonly used alteration types are leaving/removing blank lines among code lines, changing identifier names, adding/removing comment lines and replacing of code blocks of methods. In accordance with the experimental test results, doing one of these changing is not efficient while decreasing similarity ratio. If student combines all of six steps that mentioned in Section 3.3, the similarity among original code and copy code decreases significantly. However, this is difficult as creating new code, so instructors claim that students don’t exert effort and waste their time to combine more than three steps.

### *6.2.3 Comparing Results Our Tool with Other Similarity Program*

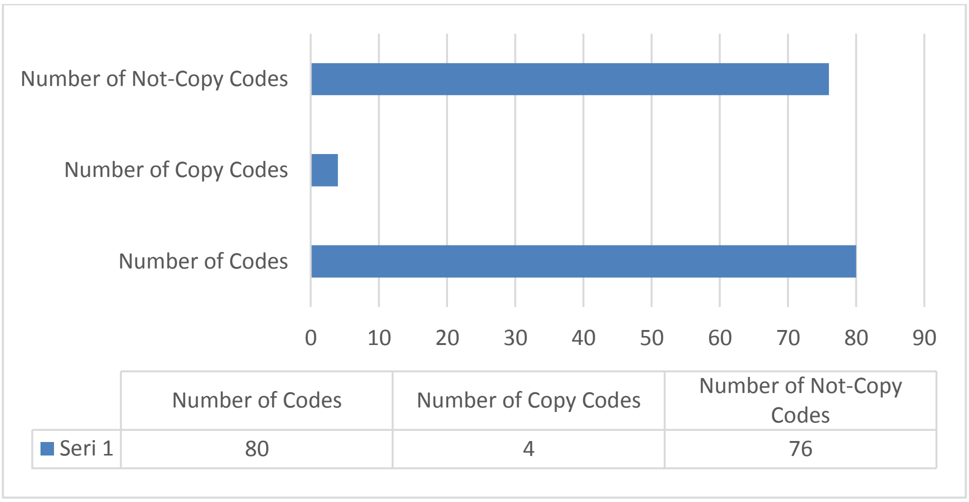
When we compare our results for one dataset with JPLAG results, we realized that the results are similar. Both of two tool extracts same copy codes. However, our tool find more copy codes according to JPLAG. When we inspect the other copy codes that is catched by out tool, we saw that similarity results are consistent. While JPLAG found 4 copy in all dataset which contains 80 assignments, our n-gram tool found 7 copy. Also JPLAG couldn't parse 4 documents because of including Turkish character, but tool developed for this study parsed all of code.

Figure 6.5 and 6.6 show the results of our tool and JPLAG for the same dataset.



| | Number of Codes | Number of Copy Codes | Number of Not-Copy Codes |
|---|---|---|---|
| ■ Seri 1 | 80 | 7 | 73 |

Figure 6.5 Similarity Results of the tool which is developed for this study



| | Number of Codes | Number of Copy Codes | Number of Not-Copy Codes |
|---|---|---|---|
| ■ Seri 1 | 80 | 4 | 76 |

Figure 6.6 Similarity Results of JPLAG

# CHAPTER SEVEN
# THE APPLICATION

In this study, a system is developed for finding similarities between source code documents. At the beginning, only N-gram algorithm is tried on two different datasets. According to results, it can be easily seen that the algorithm is so effective for not only documents natural language documents, but also source code documents. Language independence provides flexibility while evaluating programs because it doesn't construct user to choose any programming language. N-gram algorithms can work with every textual documents. So, first and second experimental results are succesfull for analyzing source code documents.

In addition to N-gram algorithm, VSM is tried on N-gram terms on other datasets that is constructed by modifying a source code in various forms. When VSM and N-gram model is combined, it is realized that similarity results are more consistent. So the study is improved by VSM in second experimental study part.

At the end of the study, a metric based system is developed. Firstly, metric types that will be used in comparing source codes are specified. Then, an interface provide to make an examination on two similar source code documents with evaluating metrics' results.

## 7.1 Substring Matching Part

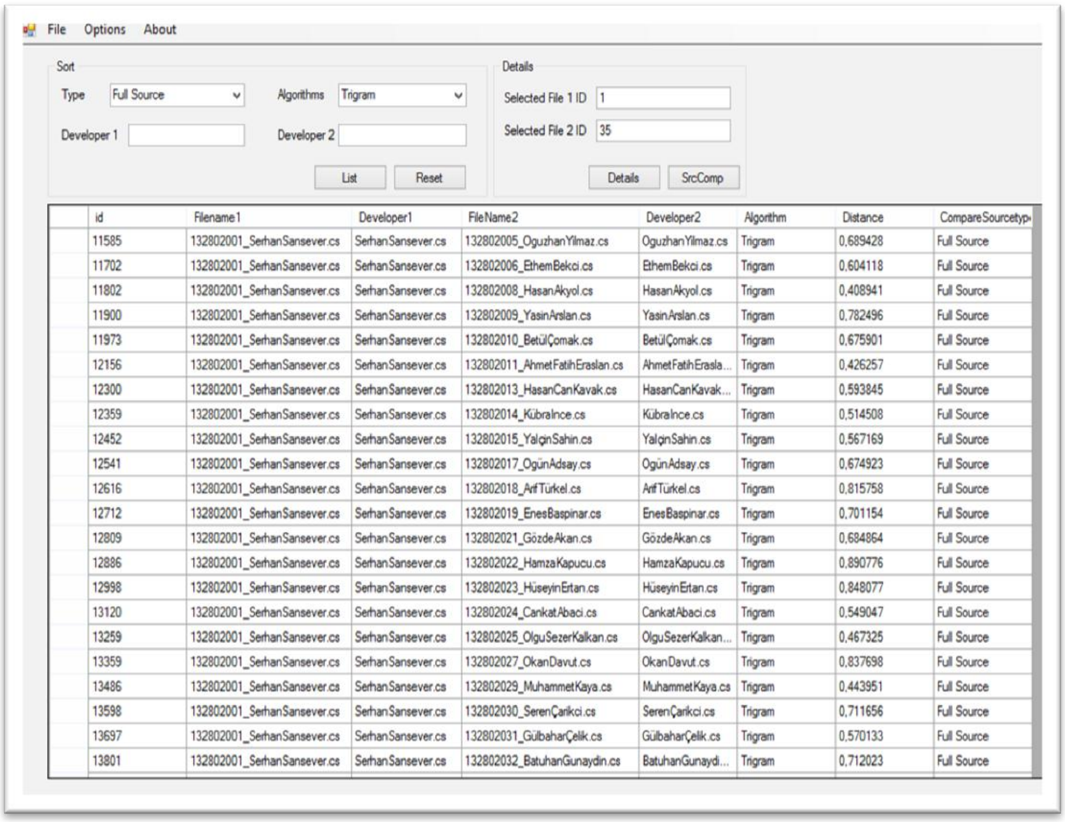
In the application both substiring matching method (N-gram algorithm) and fingerprint control method are tried on different datasets. N-gram model is choosen for providing language independence. So the base of this study depends on N-gram algorithm results.

Metric based system let only analyzing a source code that is written by constraint programming languages. So, in this study only codes written by C programming languages can be inspected by metric based method.

### *7.1.1 N-gram Results*

In this study, the base method is N-gram algorithm to analyze source codes and detecting similarities. Firstly, bi-gram and tri-gram sub-strings are obtained from the source codes in document set. According to N-gram algorithm the similarity results are revealed. According to similarity scores, it is possible to say that the program is plagiarized or not.

Figure 7.1 shows the N-gram results of dataset that is mentioned in 6.1.1.

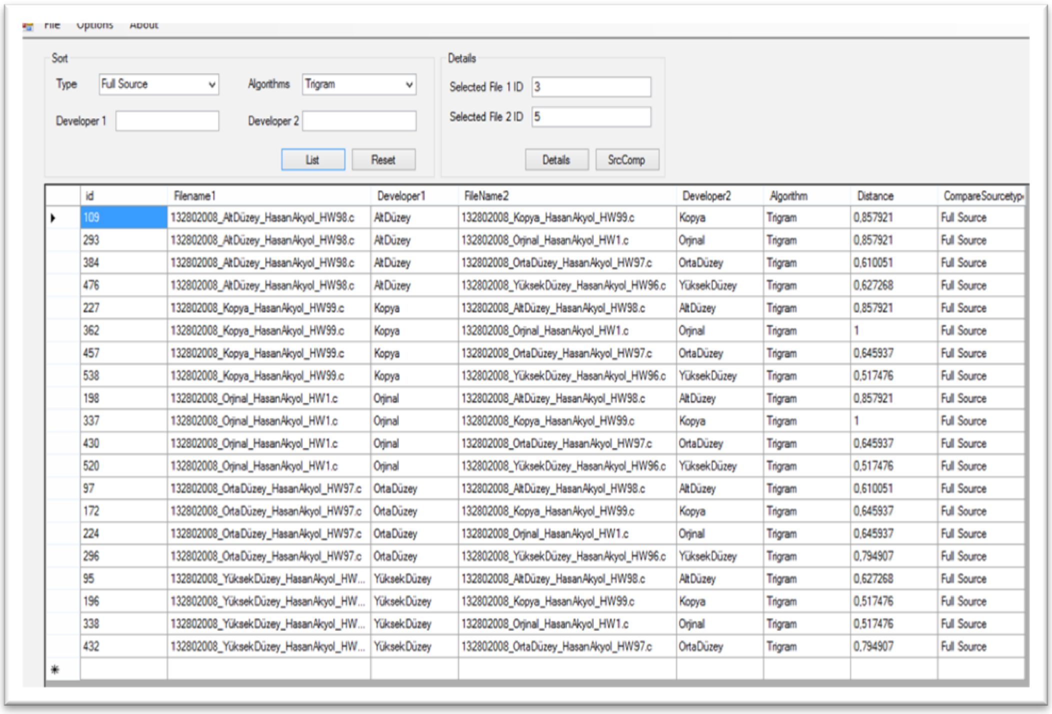


Type: Full Source
Algorithms: Trigram
Developer 1:
Developer 2:
Selected File 1 ID: 1
Selected File 2 ID: 35

| id | Filename1 | Developer1 | FileName2 | Developer2 | Algorithm | Distance | CompareSourcetyp |
|---|---|---|---|---|---|---|---|
| 11585 | 132802001_SerhanSansever.cs | SerhanSansever.cs | 132802005_OguzhanYilmaz.cs | OguzhanYilmaz.cs | Trigram | 0,689428 | Full Source |
| 11702 | 132802001_SerhanSansever.cs | SerhanSansever.cs | 132802006_EthemBekci.cs | EthemBekci.cs | Trigram | 0,604118 | Full Source |
| 11802 | 132802001_SerhanSansever.cs | SerhanSansever.cs | 132802008_HasanAkyol.cs | HasanAkyol.cs | Trigram | 0,408941 | Full Source |
| 11900 | 132802001_SerhanSansever.cs | SerhanSansever.cs | 132802009_YasinArslan.cs | YasinArslan.cs | Trigram | 0,782496 | Full Source |
| 11973 | 132802001_SerhanSansever.cs | SerhanSansever.cs | 132802010_BetülÇomak.cs | BetülÇomak.cs | Trigram | 0,675901 | Full Source |
| 12156 | 132802001_SerhanSansever.cs | SerhanSansever.cs | 132802011_AhmetFatihEraslan.cs | AhmetFatihErasla... | Trigram | 0,426257 | Full Source |
| 12300 | 132802001_SerhanSansever.cs | SerhanSansever.cs | 132802013_HasanCanKavak.cs | HasanCanKavak... | Trigram | 0,593845 | Full Source |
| 12359 | 132802001_SerhanSansever.cs | SerhanSansever.cs | 132802014_KübraInce.cs | KübraInce.cs | Trigram | 0,514508 | Full Source |
| 12452 | 132802001_SerhanSansever.cs | SerhanSansever.cs | 132802015_YalçinSahin.cs | YalçinSahin.cs | Trigram | 0,567169 | Full Source |
| 12541 | 132802001_SerhanSansever.cs | SerhanSansever.cs | 132802017_OgünAdsay.cs | OgünAdsay.cs | Trigram | 0,674923 | Full Source |
| 12616 | 132802001_SerhanSansever.cs | SerhanSansever.cs | 132802018_ArifTürkel.cs | ArifTürkel.cs | Trigram | 0,815758 | Full Source |
| 12712 | 132802001_SerhanSansever.cs | SerhanSansever.cs | 132802019_EnesBaspinar.cs | EnesBaspinar.cs | Trigram | 0,701154 | Full Source |
| 12809 | 132802001_SerhanSansever.cs | SerhanSansever.cs | 132802021_GözdeAkan.cs | GözdeAkan.cs | Trigram | 0,684864 | Full Source |
| 12886 | 132802001_SerhanSansever.cs | SerhanSansever.cs | 132802022_HamzaKapucu.cs | HamzaKapucu.cs | Trigram | 0,890776 | Full Source |
| 12998 | 132802001_SerhanSansever.cs | SerhanSansever.cs | 132802023_HüseyinErtan.cs | HüseyinErtan.cs | Trigram | 0,848077 | Full Source |
| 13120 | 132802001_SerhanSansever.cs | SerhanSansever.cs | 132802024_CankatAbaci.cs | CankatAbaci.cs | Trigram | 0,549047 | Full Source |
| 13259 | 132802001_SerhanSansever.cs | SerhanSansever.cs | 132802025_OlguSezerKalkan.cs | OlguSezerKalkan... | Trigram | 0,467325 | Full Source |
| 13359 | 132802001_SerhanSansever.cs | SerhanSansever.cs | 132802027_OkanDavut.cs | OkanDavut.cs | Trigram | 0,837698 | Full Source |
| 13486 | 132802001_SerhanSansever.cs | SerhanSansever.cs | 132802029_MuhammetKaya.cs | MuhammetKaya.cs | Trigram | 0,443951 | Full Source |
| 13598 | 132802001_SerhanSansever.cs | SerhanSansever.cs | 132802030_SerenÇarikci.cs | SerenÇarikci.cs | Trigram | 0,711656 | Full Source |
| 13697 | 132802001_SerhanSansever.cs | SerhanSansever.cs | 132802031_GülbaharÇelik.cs | GülbaharÇelik.cs | Trigram | 0,570133 | Full Source |
| 13801 | 132802001_SerhanSansever.cs | SerhanSansever.cs | 132802032_BatuhanGunaydin.cs | BatuhanGunaydi... | Trigram | 0,712023 | Full Source |

Figure 7.1 Similarity scores in dataset which consists of students assignments

Figure 7.2 shows the N-gram results of dataset that is mentioned in 6.1.2.



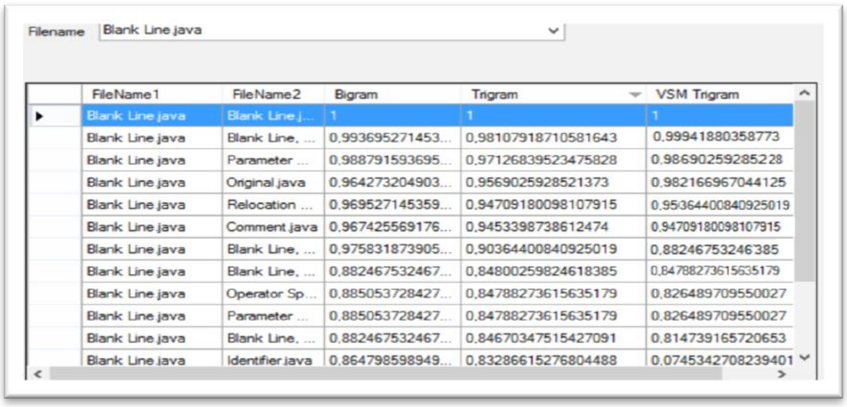
Type: Full Source
Algorithms: Trigram
Developer 1:
Developer 2:
Selected File 1 ID: 3
Selected File 2 ID: 5

| id | Filename1 | Developer1 | FileName2 | Developer2 | Algorithm | Distance | CompareSourcetyp |
|---|---|---|---|---|---|---|---|
| 109 | 132802008_AltDüzey_HasanAkyol_HW98.c | AltDüzey | 132802008_Kopya_HasanAkyol_HW99.c | Kopya | Trigram | 0,857921 | Full Source |
| 293 | 132802008_AltDüzey_HasanAkyol_HW98.c | AltDüzey | 132802008_Orjinal_HasanAkyol_HW1.c | Orjinal | Trigram | 0,857921 | Full Source |
| 384 | 132802008_AltDüzey_HasanAkyol_HW98.c | AltDüzey | 132802008_OrtaDüzey_HasanAkyol_HW97.c | OrtaDüzey | Trigram | 0,610051 | Full Source |
| 476 | 132802008_AltDüzey_HasanAkyol_HW98.c | AltDüzey | 132802008_YüksekDüzey_HasanAkyol_HW96.c | YüksekDüzey | Trigram | 0,627268 | Full Source |
| 227 | 132802008_Kopya_HasanAkyol_HW99.c | Kopya | 132802008_AltDüzey_HasanAkyol_HW98.c | AltDüzey | Trigram | 0,857921 | Full Source |
| 362 | 132802008_Kopya_HasanAkyol_HW99.c | Kopya | 132802008_Orjinal_HasanAkyol_HW1.c | Orjinal | Trigram | 1 | Full Source |
| 457 | 132802008_Kopya_HasanAkyol_HW99.c | Kopya | 132802008_OrtaDüzey_HasanAkyol_HW97.c | OrtaDüzey | Trigram | 0,645937 | Full Source |
| 538 | 132802008_Kopya_HasanAkyol_HW99.c | Kopya | 132802008_YüksekDüzey_HasanAkyol_HW96.c | YüksekDüzey | Trigram | 0,517476 | Full Source |
| 198 | 132802008_Orjinal_HasanAkyol_HW1.c | Orjinal | 132802008_AltDüzey_HasanAkyol_HW98.c | AltDüzey | Trigram | 0,857921 | Full Source |
| 337 | 132802008_Orjinal_HasanAkyol_HW1.c | Orjinal | 132802008_Kopya_HasanAkyol_HW99.c | Kopya | Trigram | 1 | Full Source |
| 430 | 132802008_Orjinal_HasanAkyol_HW1.c | Orjinal | 132802008_OrtaDüzey_HasanAkyol_HW97.c | OrtaDüzey | Trigram | 0,645937 | Full Source |
| 520 | 132802008_Orjinal_HasanAkyol_HW1.c | Orjinal | 132802008_YüksekDüzey_HasanAkyol_HW96.c | YüksekDüzey | Trigram | 0,517476 | Full Source |
| 97 | 132802008_OrtaDüzey_HasanAkyol_HW97.c | OrtaDüzey | 132802008_AltDüzey_HasanAkyol_HW98.c | AltDüzey | Trigram | 0,610051 | Full Source |
| 172 | 132802008_OrtaDüzey_HasanAkyol_HW97.c | OrtaDüzey | 132802008_Kopya_HasanAkyol_HW99.c | Kopya | Trigram | 0,645937 | Full Source |
| 224 | 132802008_OrtaDüzey_HasanAkyol_HW97.c | OrtaDüzey | 132802008_Orjinal_HasanAkyol_HW1.c | Orjinal | Trigram | 0,645937 | Full Source |
| 296 | 132802008_OrtaDüzey_HasanAkyol_HW97.c | OrtaDüzey | 132802008_YüksekDüzey_HasanAkyol_HW96.c | YüksekDüzey | Trigram | 0,794907 | Full Source |
| 95 | 132802008_YüksekDüzey_HasanAkyol_HW... | YüksekDüzey | 132802008_AltDüzey_HasanAkyol_HW98.c | AltDüzey | Trigram | 0,627268 | Full Source |
| 196 | 132802008_YüksekDüzey_HasanAkyol_HW... | YüksekDüzey | 132802008_Kopya_HasanAkyol_HW99.c | Kopya | Trigram | 0,517476 | Full Source |
| 338 | 132802008_YüksekDüzey_HasanAkyol_HW... | YüksekDüzey | 132802008_Orjinal_HasanAkyol_HW1.c | Orjinal | Trigram | 0,517476 | Full Source |
| 432 | 132802008_YüksekDüzey_HasanAkyol_HW... | YüksekDüzey | 132802008_OrtaDüzey_HasanAkyol_HW97.c | OrtaDüzey | Trigram | 0,794907 | Full Source |

Figure 7.2 Similarity scores in dataset which consists of five modified codes

### *7.1.2 VSM N-gram Results*

In this part of study, N-gram results of each documents are seperately stored in document term matrices in VSM. Acording to observations of instructors that place into our study, VSM N-gram results are more reliable. In Figure 8.3 shows similarity ratios between Blank Line document and other documents.



Filename: Blank Line.java

| FileName1 | FileName2 | Bigram | Trigram | VSM Trigram |
|---|---|---|---|---|
| Blank Line.java | Blank Line.j... | 1 | 1 | 1 |
| Blank Line.java | Blank Line, ... | 0,993695271453... | 0,98107918710581643 | 0,99941880358773 |
| Blank Line.java | Parameter ... | 0,988791593695... | 0,97126839523475828 | 0,98690259285228 |
| Blank Line.java | Original.java | 0,964273204903... | 0,9569025928521373 | 0,982166967044125 |
| Blank Line.java | Relocation ... | 0,969527145359... | 0,94709180098107915 | 0,95364400840925019 |
| Blank Line.java | Comment.java | 0,967425569176... | 0,9453398738612474 | 0,94709180098107915 |
| Blank Line.java | Blank Line, ... | 0,975831873905... | 0,90364400840925019 | 0,88246753246385 |
| Blank Line.java | Blank Line, ... | 0,882467532467... | 0,84800259824618385 | 0,84788273615635179 |
| Blank Line.java | Operator Sp... | 0,885053728427... | 0,84788273615635179 | 0,826489709550027 |
| Blank Line.java | Parameter ... | 0,885053728427... | 0,84788273615635179 | 0,826489709550027 |
| Blank Line.java | Blank Line, ... | 0,882467532467... | 0,84670347515427091 | 0,814739165720653 |
| Blank Line.java | Identifier.java | 0,864798598949... | 0,83286615276804488 | 0,0745342708239401 |

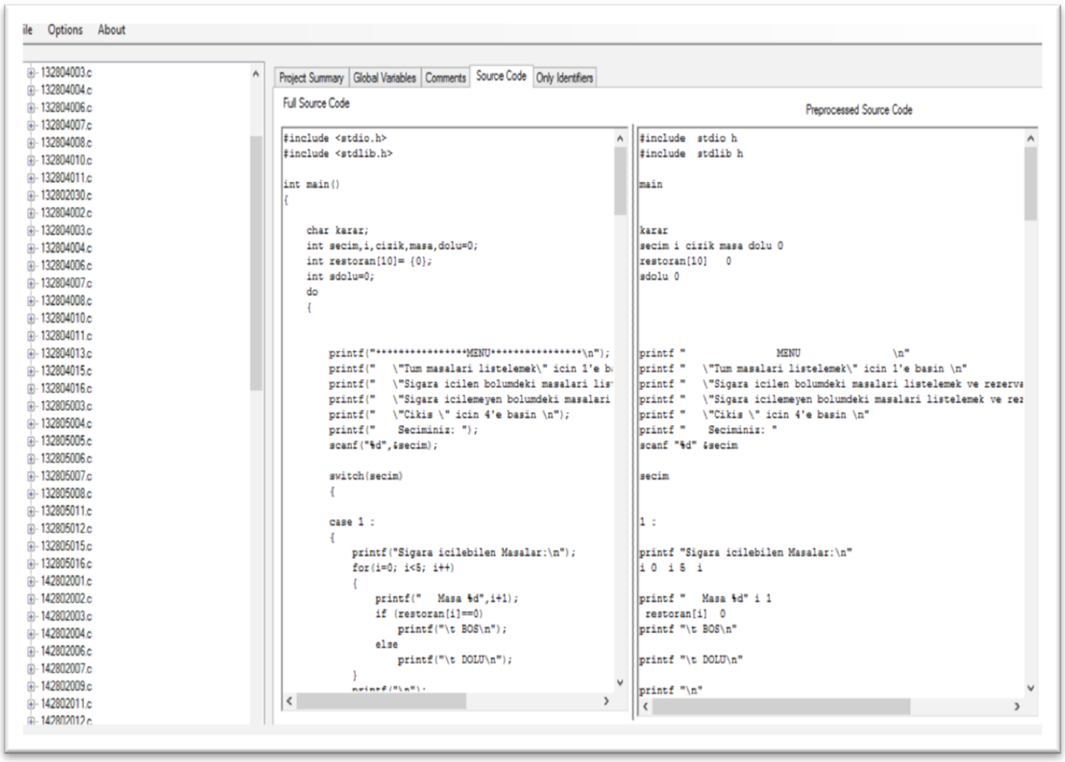
Figure 7.3 Bi-gram, tri-gram and VSM tri-gram results interface

## 7.2 Fingerprint System Part

### *7.2.1 Metric Based System*

In this part of study, distinctive software metrics in source codes are determined firstly. The metrics that are choosen for this study;

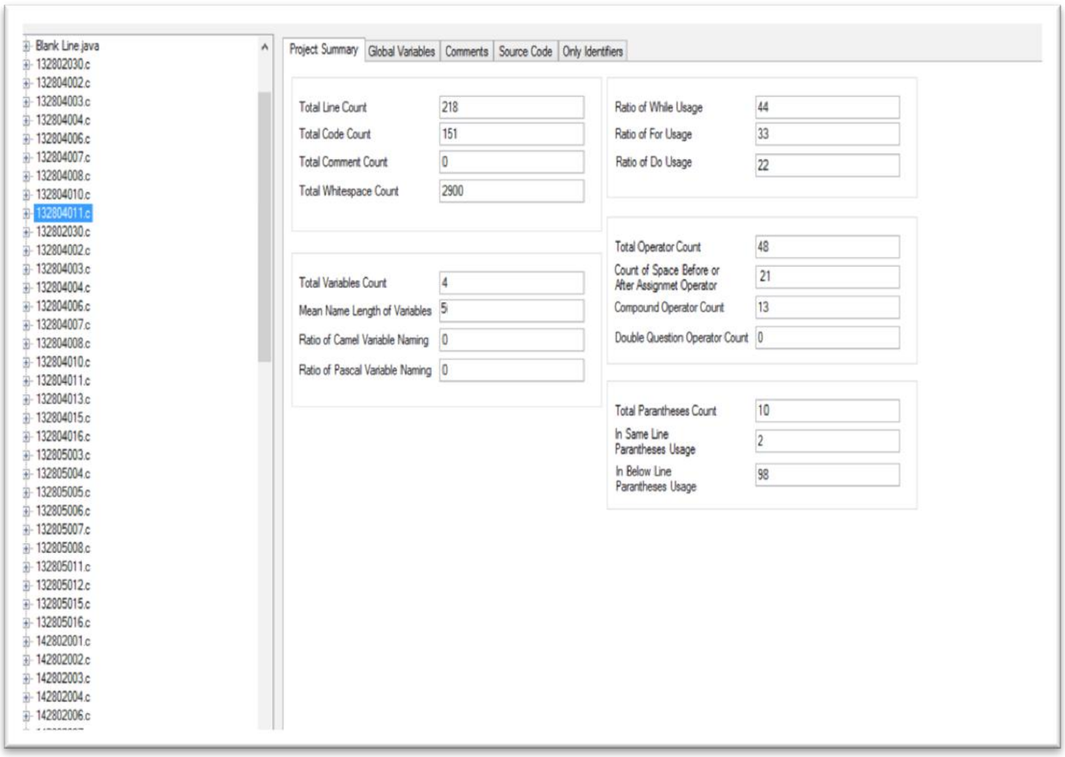
- Total lines count( including blank lines),
- Total code lines count,
- Total comment lines count,
- Total variables count,
- Mean length of variable names,
- Ratio of camel variable naming,
- Ratio of pascal variable naming,
- Ratio of while usage,
- Ratio of for usage,
- Ratio of do-while usage,
- Total operator number count (+, -, *, =, <, >, <= etc…),
- Count of space before or after assignment operator,
- Compound operator count (++,--, ==, etc…),
- Double question operators count (??),
- Total curly bracket count,
- Percentage of usage of curly brackets in same line,
- Percentage of usage of curly brackets in below line.



```
#include <stdio.h>
#include <stdlib.h>

int main()
{

    char karar;
    int secim,i,cizik,masa,dolu=0;
    int restoran[10]= {0};
    int sdolu=0;
    do
    {


        printf("***************MENU***************\n");
        printf("   \"Tum masalari listelemek\" icin 1'e b
        printf("   \"Sigara icilen bolumdeki masalari lis
        printf("   \"Sigara icilemeyen bolumdeki masalari
        printf("   \"Cikis \" icin 4'e basin \n");
        printf("    Seciminiz: ");
        scanf("%d",&secim);

        switch(secim)
        {

        case 1 :
        {
            printf("Sigara icilebilen Masalar:\n");
            for(i=0; i<5; i++)
            {
                printf("   Masa %d",i+1);
                if (restoran[i]==0)
                    printf("\t BOS\n");
                else
                    printf("\t DOLU\n");
            }
```

```
#include  stdio h
#include  stdlib h

main


karar
secim i cizik masa dolu 0
restoran[10]   0
sdolu 0




printf "               MENU               \n"
printf "   \"Tum masalari listelemek\" icin 1'e basin \n"
printf "   \"Sigara icilen bolumdeki masalari listelemek ve rezerva
printf "   \"Sigara icilemeyen bolumdeki masalari listelemek ve rez
printf "   \"Cikis \" icin 4'e basin \n"
printf "    Seciminiz: "
scanf "%d" &secim

secim


1 :

printf "Sigara icilebilen Masalar:\n"
i 0  i 5  i

printf "   Masa %d" i 1
 restoran[i]  0
printf "\t BOS\n"

printf "\t DOLU\n"

printf "\n"
```

Figure 7.4 One of screenshots of metric based system



| | |
|---|---|
| Total Line Count | 218 |
| Total Code Count | 151 |
| Total Comment Count | 0 |
| Total Whitespace Count | 2900 |
| Total Variables Count | 4 |
| Mean Name Length of Variables | 5 |
| Ratio of Camel Variable Naming | 0 |
| Ratio of Pascal Variable Naming | 0 |
| Ratio of While Usage | 44 |
| Ratio of For Usage | 33 |
| Ratio of Do Usage | 22 |
| Total Operator Count | 48 |
| Count of Space Before or After Assignmet Operator | 21 |
| Compound Operator Count | 13 |
| Double Question Operator Count | 0 |
| Total Parantheses Count | 10 |
| In Same Line Parantheses Usage | 2 |
| In Below Line Parantheses Usage | 98 |

Figure7.5 Value of metrics in a code in dataset

# CHAPTER EIGHT
# CONCLUSION AND FUTURE WORK

## 8.1 Conclusion

In this study, we analyzed source code documents and try to detect similarities among them. In accordance with this purpose, we developed an application that can work on documents written in any programming language. So, when we search the literature, we understood that N-gram algorithm is one of the best alternatives to reach our aim.

Mainly, the application has three separate parts. These parts are formed with different datasets and methods.

In fact, core of our study is N-gram algorithm. So, we began with performing N-gram algorithm on two different datasets. Experiments on first dataset showed us that N-gram algorithm can truly categorize documents according to similarity results. Also, we observed that how N-gram algorithm efficient in programs written in programming language. When N-gram results were indicated, we saw that documents of C and C# programming languages are grouped separately. When test process was completed, results of first part of application demonstrated that our application did its duty successfully.

In second part, we want to enhance our study results. So, we tried to combine VSM with N-gram algorithm. After obtaining bi-gram and tri-gram terms from each documents, they were placed into document term matrices. The distances of these matrices from the each other were counted with using CN method and similarity ratios are got. When VSM tri-gram results and tri-gram results were compared, it seems pretty obvious that VSM tri-gram results are better than tri-gram results to detect similarities.

Finally, we tried metric based system. In fact, this system can work with only specified programming language. For example, if a metric based system is developed for testing documents written by C Programming Language, it may not work on documents that is created using Java Programming Language. So, we tried only C source code documents to test metric based system. Before, we look at previous studies in literature and we decided to software metrics that are generally are used in finding similarities between source codes. We use this system for only checking whose metrics are similar to each other. That is, this system doesn’t return any similarity score, it enables inspection.

After all, it can be said that the application of this study attain our goal successfully. In each step, it can be realized that we make our study better.

## 8.2 Future Work

Plagiarism in programming courses is a growing problem in education. The aim of this study is attempting to find similar source codes. Although N-gram analysis is a well-known technique in NLP, it has been utilizing in source code analysis for a while. In this study, the reason of selecting N-gram method is providing language independency. In this study, firstly, we utilized only bi-gram and tri-gram methods to check the source codes are copy or not. Also, additional to n-grams, VSM is constructed and weights of tri-grams of all documents are placed into document matrices separately. Then, CSM scores are obtained between matrices in VSM. When acquired tri-gram and VSM tri-gram results are compared, it is determined by instructors that VSM tri-gram results give more accurate outcomes. Also, we touched on fingerprint based system with calculating some specific metric values.

In future, we would like to integrate Word Net to our proposed method that provides finding similarities among source code. Another future direction of proposed study is generating a system that enables to build up greater datasets to test the study.

# REFERENCES

Aiken, A. (2014). *Moss: A system for detecting software plagiarism*. Retrieved (May 25, 2015) from www.theory.stanford.edu/~aiken/moss/

Bozyigit, F., Kılınç, D., Kut, A. and Kaya, M. (2015). Bulanık mantık algoritmaları kullanarak kaynak kod benzerliği bulma. *Akademik Bilişim 2015* (in press).

Brocardo, M. L., Traore, I. and Saad, S., Woungang, I. (2013). Authorship verification for short messages using stylometry. *Conference on Computer, Information and Telecommunication Systems (CITS).*

Cavnar, W. and Trenkle, J. (1994). N-gram-based text categorization. *3rd Annual Symposium on Document Analysis and Information Retrieval SDAIR-94.*

Chen, X., Francia, B., Li, M., McKinnon, B. and Seker, A. (2004). Shared information and program plagiarism detection. *IEEE Transaction Engineering Education*, 50(7), 1545-1551.

Cosma, G. (2008). *An approach to source-code plagiarism detection and investigation using latent semantic analysis*. PhD Thesis, University of Warwick, UK.

Culwin, F. and Lancaster, T. (2001). Plagiarism issues for higher education. *VINE*, 31(2), 36 – 41.

Dale, R., Mois, H. and Somers, H. (2000). *Handbook of NLP*. New York: Marcel Dekker.

Ding, H. and Samadzadeh, M. H. (2004). Extraction of java program fingerprints for software authorship identification. *The Journal of Systems and Software*, 72(1), 49-57.

Fan, W., Wallace, L., Rich, S. and Zhang, Z. (2006). Tapping into the power of text mining. *Communications of ACM*, 49(9), 76-82.

Feldman, R. and Dagan, I. (1995). Kdt - knowledge discovery in texts. In *Proceedings of the First International Conference on Knowledge Discovery (KDD)*, 112–117.

Frantzeskou, G., Stamatatos, E., Gritzalis, S. and Chaski, C. E. (2007). Identifying authorship by byte-level n-grams: the source code author profile (SCAP). *Journal of Digital Evidence,* 6(1), 508-515.

Frantzeskou, G., Gritzalis, E., Stamatatos, E. and Katsikas, S. (2006). Effective identification of source code authors by using byte-level informations. *Artificial Intelligence and Innovations,* 204, 508-515.

Gray, A., Sallis, P. and Macdonell, S. (1997). Software forensics: extending authorship analysis techniques to computer programs. *Proceedings of the 3rd Biannual Conference International Association of Forensic Linguists (IAFL'97).*

Jones, E. L. (2001). Metrics based plagiarism monitoring. *6th Annual CCSC Northeastern Conference,* Middlebury, Vermont, 20-21.

Joy, M. and Luck, M. (1999). Plagiarism in programming assignments. *IEEE Trans. Educ*., 42(1), 129–133.

Kaohsiung, T. (2010). Computational Collective Intelligence. *Second International Conference, ICCCI 2010*.

Keselj, P. and Thomas, C. (2003). N-gram based author profiles for authorship attribution. *Proceedings of the Conference Pacific Association for Computational Linguistics, PACLING03*, 255-264.

Khreisat, L. (2006). Arabic text classification using N-gram frequency statistics a comparative study. *Proceedings of the International Conference on Data Mining (DMIN2006),* Las Vegas, USA, 78-82.

Kılınç, D., Bozyiğit, F., Kut, A. and Kaya, M. (2015). Overview of source code plagiarism in programming courses. *International Journal of Soft Computing and Engineering (IJSCE),* 5(2), 79-85.

Krsul, I., and Spafford, E. (1997). Authorship analysis: identifying the author of a program. *Computers and Security*, 16(3), 233-248.

Malpohl, G. (2006). *JPlag: Detecting software plagiarism*. Retrieved May 25, 2015 from http://www.ipd.uka.de:2222/index.html

Manning, C. D., Raghavan, P. and Schütze, H. (2009). *An introduction to information retrieval*. Cambridge, England: Cambridge University Press.

Martin, B. (1994). Plagiarism: a misplaced emphasis. *Journal of Information Ethics,* 3(2), 36–47.

Parker, A. and Hamblen, J. (1989). Computer algorithms for plagiarism detection. *IEEE Transaction Engineering Education*, 32, 94–99.

Salton, G. (1971). *The SMART retrieval system - experiments in automatic document processing.* NJ, Englewood Cliffs: Prentice-Hall.

Salton, G., Wong, A. and Yang, C. S. (1975). A vector space model for information retrieval. *Journal of the American Society for Information Science*, 18(11), 613-620.

Schleimer, S., Wilkerson, D. and Aiken, A. (2003). Winnowing: local algorithms for document fingerprinting. *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, 76–85, New York, NY, USA.

Whale, G. (1988). Plague: plagiarism detection using program structure. *Department of Computer Science Technical Report 8805*, University of NSW, Kensington, Australia.

Wise, M. (1996). Yap3: improved detection of similarities in computer program and other texts. *SIGCSE Bulletin*, 28(1), 130–134.

*Vector space model*. (2009). Retrieved May 25, 2015 from http://nlp.stanford.edu/IR-book/html/htmledition/the-vector-space-model-for-scoring-1.html