International Journal of Computer Applications (0975 – 8887)
Volume 66– No.4, March 2013

Author Identification: An Approach based on Code Feature Metrics using Decision Trees

Rohit R. Joshi
Computer Science Department,
Walchand Institute of
Technology, Solapur, India.

Rajesh V. Argiddi
Associate Prof.
Computer Science Department,
Walchand Institute of
Technology,Solapur, India.

Sulabha S. Apte, PhD.
Professor
Computer Science Department,
Walchand Institute of
Technology,Solapur, India.

ABSTRACT
Now a day's, cases of piracies, copyrights, legal disputes, and allegations are increasing as far as field of software is concerned. In such cases it is difficult to say who is right and who is wrong between the two quarreling parties due to insufficient evidences. So, what could be done in such cases is the question? The field called as software forensic can help in such cases by giving the right direction towards the case. Software forensic is the field which can analyze the code from different viewpoints and helps in extracting the code metrics which can be syntactic, semantic, structural, behavioral, stylometric etc. These metrics can help in doing author identification, discrimination, characterization etc. Author identification plays very important role in most of the cases such as plagiarism detection, masquerade detection, software maintainability and resolving authorship disputes. This paper focuses on author identification, source code metrics, related work, proposed work and applications of author identification.

General Terms
Software Forensics, Decision trees, Authorship analysis

Keywords
Software Source Code Metrics., Author Identification, Plagiarism Detection

1. INTRODUCTION
As mentioned above software forensic allow to study and analyze the source code in many different ways. It may be syntactical, structural, and behavioral. MacDonell S.G. et al [1] have discussed following four applications of software forensic:

1: Author Identification

2: Author Characterization

3: Author Discrimination

4: Author Intent Determination

1.1 Author Identification:
As mentioned above, author identification involves the likelihood identification of the author of a given code. The principle here is to take a piece of code to be identified and extract the characteristics of that author (stylistic, structural, behavioral etc.) out of that code and then match those characteristics with the already existing characteristics of the same author. So, must condition here is to have the previous records of the authors to be tested.

1.2 Author Discrimination:
It involves making distinction between two or more authors if they have written the same code together. In some cases, a module can be divided in to two and given to two different authors for completion. Afterwards, the module is again combined to have a complete full module. So, in this case a source code can have more than two authors. Identifying this is nothing but author discrimination.

1.3 Author Characterization:
It involves characterizing the author instead of identifying it. It means from the programming styles of the author, identification of the personality, educational background, technical excellence, field of expertise and manner of specifying the things etc. can be done.

1.4 Author Intent Determination:
While executing the programs, sometimes it gives erroneous results with some undesired output. Author intent determination involves detecting whether the undesired output comes really due to erroneous code or it comes due to an intentional malicious code.

Amongst all the mentioned applications above author identification is the most powerful application. Here after, the focus will be on author identification and its related things throughout this paper. There are several situations where it is necessary to identify the author of the code. The situations may involve plagiarism detection, masquerade detection, legal allegations, copyrights disputes, piracy etc. In educational fields, it is quite often in case of assignments. Students copy the codes from some source and submit as their own work. There may be disputes between organizations for some copyrighted software products. Also, anti-virus making organizations can benefit from author identification. If they have identified the author who has written the virus code, then they can find the solution to that virus very fast, from the previous results of that author. So, the question is how to identify the author from the code? Every individual has its own set of style to perform a task. For instance, an example can be taken of handwriting of a person whose style cannot change. So, by verifying the handwriting of a particular person that person can be identified. This is often used in legal document verification to avoid the frauds. Similarly, it is necessary in the field of software codes to identify the author of the codes to avoid the frauds such as software theft etc. Programming language allows programmer to write the programs in his/her own style following the standard grammar of the language. Due to this distinguish can be made between the two authors of a program and ultimately the codes of theirs. This can be done by capturing the styles of different authors and testing them against the pre-captured ones. These styles that are captured are called as source code metrics. But, is it as simple as that? Let's see this in the further sections.

2. RELATED WORK:

2.1 Source Code Metrics:

This section discusses about different software source code metrics and the work carried out by different people in this field. We know what the basic term "metrics" means? It's the measurement of a particular thing taken at a time. So, the term software source code metrics means different measurements taken of a piece of a code of software. MacDonell S.G. et al [1] have categorized software code metrics into following three types as:

1. Layout metrics
2. Style metrics
3. Structure metrics

2.1.1 *Layout Metrics:*

These metrics are extracted from look and feel viewpoint of a code. Every author of a program has its own way of presenting a program. Some authors write the code with proper margins, good indentations, good spacing etc. It's intentional so that looking at a code, one can understand what the code is about, which are different functionalities involved in the code and what they are doing? Also, a good indentation always helps in the case of nesting. Following are some layout metrics used previously:

a. *White spaces***:** This metric deal with total no. of spaces, proportion of spaces on both sides of operators, proportion of spaces on either side of operator, leading spaces, trailing spaces etc.
b. *Characters per line:* This metric deal with no. of characters per line. It helps in good view of a too long line of a code.
c. *Tabs:* This metric deal with no. of tabs either leading, trailing or inline tabs.
d. *Brace positions:* This metric deal with positions of curly braces such as braces on the same line of the statement or below the statement. It also helps in identifying whether a statement has opening and closing braces or is single statement.

2.1.2 *Style Metrics:*

These metrics are extracted from the viewpoint of the writing styles of the authors. Here, lots of variety can be expected from the authors with their styles. Some authors may try to write the code in some fixed no. of lines, while others take more no. of lines for the same code. Also, some authors have habit of writing the comments for the explanation of the code. Some authors may take some specific variable names each time for the same functionality, while others put the restriction on length of variable names used. Following are some style metric enlisted:

a. *Capital and small letters:* This metric capture the style of author of using capital or small letters in variables or method names.
b. *Lines of code (LOC):* This metric counts total no. of lines in the code. This metric may vary from technique to technique. Some takes it with white spaces, some without white spaces while some takes it with comments and some takes it without considering comments.
c. *Words per line:* This metric deal with the specific no. of words in a particular line. It makes sense in breaking the same statement and writing it in multiple lines for a clear view.
d. *Variables per methods:* This metric deal with the no. of variables used by an author per method. Many authors have the habit of doing the task in minimum no. of variables and that depends on the level of expertise of the author, whether he is beginner, intermediate or an expert.

2.1.3 *Structure Metrics:*

These metrics concerned with the structure of the program. These include looping and control statements. Every author has to use the conditional statements in his code. Some may use while; some use for, some use do-while as far as looping is concerned. Also, conditional statements such as if, else-if, switch and turnery operator statements can be used according to need. Following is the list of some general structure metrics:

a. *If-else statements:* This deals with the no. of if-else statements in the code. Here, nesting of these statements are also taken into consideration for extracting these metrics. In some cases only if statement is used, that metric should also be taken into consideration for checking of single if statements.
b. *Switch statements:* These statements can also be used by authors for decision making and usually used in case of menu driven programs. So, the no. of switch statements used is the metric considered here.
c. *For, while, do-while looping statements:* These statements are also the substitutes for one another by providing the same functions. It's on programmer which statement he wants to use to achieve the current task. So , the no. of for, while or do-while statements can be individually taken as separate metrics or groping of them together can be considered as one single metric i.e. looping metric.
d. *Conditional Operators:* It is especially concerned with the "?:" operator. Many people for single line condition checking uses this one.

So, grouping these above metrics together constitutes a full set of metrics to be evaluated from a given piece of a software source code. Some of the metrics from above listing can be treated as a Boolean metric i.e. the presence of them are only considered instead of their count. These metrics plays very important role in creating author profiles which can be stored and are used to match it with metrics extracted from the piece of code whose author is to be found.

2.2 Technologies Used

This section presents the technologies used in previous works either for extracting software source code metrics or for the author identification. They are as follows:

2.2.1 *Probabilistic Approach:*

Jay Kothari et al [2] used the approach of probability for identifying an author. Here they have considered the probability of the metric, say metric x, in to consideration , such that the metric 'x' will be classified to class/author 'i'.

For that they have taken two terms into consideration:

a. Individual Consistency: It's the measure of consistency of author to use the particular metric

b. Population Consistency: It's the measure of consistency of the metric used by number of authors.

By getting these two values, they calculated the selection criterion and applied the classification tools such as bays and VFI to classify the author of the unknown code.

2.2.2 Genetic Algorithm:

R. A. Vivanco and N. J. Pizzi[3] used genetic algorithm for identifying the effective metrics. Here, they have identified the source code metrics with the help of genetic algorithm. We know that, in GA there is concept called genes which forms the population. They have represented genes as strings of bits, with 0 representing off bit, while 1 representing on bit. Each such a bit represents metric to be used with the classifier. So, all on bits represents metrics to be used with the classifier, while all off bits represents metrics that are excluded to be used with classifier. Each gene is a set of chromosomes. Two genes acting as a parent can come together to form a child. For this, fittest genes are chosen by the function called as fitness function and the process is called as mutation. Only fittest genes are populated to the next generation.

2.2.3 N-gram Approach:

Georgia Frantzeskou et al [4] use the approach of N-gram author profile formation for author identification. In this approach they have used the concept of n-gram. N-gram is the contagious sequence that can be defined on byte, character or word. They have made an n-gram table in which it consists of n-grams found for a particular file along with its corresponding frequency. The author source codes are then combined into a single big file per author and set of L-most frequent n-grams are extracted from it. This is how the author profiles are built. For testing a file for author identification they have built a test profile from the test file. This profile is then tested against the existing author profiles to classify the test file to some author. The algorithms such as nearest neighbor or similarity search are used.

2.2.4 Neural Network, Discriminant Analysis and Case Based Reasoning:

Frantzeskou G et al [5] discussed three different approaches as follows:

a. Neural Network: This is very good technology consisting of multiple layers of nodes one following the other. The popular technique in neural network is Feed-Forward neural network especially used with back propagation. They have good learning, adapting and generalizing capabilities.

b. Discriminant Analysis: It is a statistical technique that operates with continuous variable measurements. It applies those measurements to different sets of elements to distinguish the sets. From these measurements one can classify the new elements.

c. *Case Based Reasoning:* This system is based on analogical reasoning. The results of previous cases are taken in to consideration and are analyzed to get help to give the solution to the current case. There are 4 stages in CBR as follows:

1. *Retrieval:* similar cases are retrieved to think on the current case

2. *Reuse:* The solutions of the retrieved cases can be reused to form the solution to the current case

3. *Revision:* The solution of the current case can be revised

4. *Retention:* The retention of the solution of the current case to the repository

Table 1: Comparative study of existing techniques

Approach	**Technique**	**No. of Authors**	**Total no. of samples**	**No. of training samples**	**No. of testing samples**	**Result %**
Probabilistic approach [2]	Bays /VFI	12	2110	1287	823	61% / 76% [Bays / VFI]
GA[3]	LDA	-	338	-	-	62.7%

Above table (see Table 1) discusses about the comparative study of different techniques mentioned earlier. Jay Kothari et al [2] have taken in total 2110 samples out of which 1287 are for training and 823 for testing purpose. They have considered these samples from open source projects and got 61% correct classification of unidentified samples using Bays classifier, while got 76% of correct classification using VFI classifier. Also, R. A. Vivanco and N. J. Pizzi[3] have used GA technique along with LDA algorithm for the identification of effective metrics. They have also used the testing method called leave-one-out validation. They have got 62.7% as the classification result for 338 samples. The aim of the proposed work with the decision tree technique is to take 5-8 no. of authors approximately with around 1000-1500 code samples in total.

3. PROPOSED WORK

3.1 Software Source Code Metrics:

We have seen that different metrics have been used in the literature and are also classified accordingly. The metrics corresponding to java language for proposed study are as follows:

1. *Leading Spaces:* These are the spaces given at the start of each line. Many programmers have this habit. Especially, target authors are beginners who many times gives such kind of spaces e.g. for good indentation they forget to give a tab and instead gives 4/8 spaces. This is the counting type of metric.

2. *Trailing Spaces:* These are the spaces that are present at the end of the line unknowingly. This is the counting type of metric.

3. *Trailing Tabs:* There are the tabs at the end of the line unknowingly remained by some of the authors. This is the counting type of metric.

4. *Leading Tabs:* These are the tabs at the start of each line. This is an intentional kind of thing done by good authors for the purpose of better representation of a program. This is the counting type of metric.

5. *i-as-iterator:* There is the natural tendency of some authors to use variable 'i' as iterator in every loop they use. Some authors intentionally avoid it or some authors while doing nesting at least use it once. Here, only the presence or absence of this metric can be checked and not taking the count of it. Because, counting no. of i's from the program on per author basis is simply useless.

6. Line-Length: This metric deal with the no. of characters per line. Some author has the habit of writing the statement of the code into single line while some other authors write it in multiple lines for better understanding and representation of the code. This is the counting type of metric.

7. Lines-of-Code: This metric deal with no. of lines of code on per author basis. Some expertise authors have the habit of writing the code in some specific no. of lines only due to good logical skills of theirs. While beginners don't bother about the lines of code of a program or logic even. They only have to perform the task in front of them. So, this makes the difference between the authors using this metric. This is the counting type of metric.

8. Brace Position: This metric covers the curly bracket positions. It includes the proper opening and closing of the curly braces. Also, the presence of curly braces at different positions such as at the end of the statement immediately, at the immediate next line of the end of the statement, both opening and closing curly braces on the same line of the end of the statement, both the opening and closing curly brackets appears in the same column. This is the counting type of metric.

9. Comments: This metric deal with comments used by the authors for the description of the code they have written. We have taken comments as the overall and not differentiating as large or small comments. No. of single line comments and multiline comments or their presence or absence is taken into consideration. This is the counting type of metric.

10. Average Procedure Length: This metric deal with the average no. of lines per method. This is the counting type of metric and taken on "per class basis" for each author.

11. Conditional Statements: This metric only deal with the presence or absence of the conditional statement. The conditional statement means here the turnery operator"?:" This is not the counting type of metric , instead it is Boolean type of metric.

12. Average Indentations: These are average indentations of an author taken on "per class basis / per file basis". In this, indentations with respect to leading characters are taken into consideration.

13. No. of Methods: This metric deal with no. of methods on "per class basis" for each author. It is quite often for the authors that they include a method in their code. But, for which functionality to use the method and for which to not depend on individual author and that makes the difference in the two authors. This is the counting type of metric.

14. Try Statements: This metric deal with the no. of try statements used. This is the counting type of metric as well as Boolean.

15. Unary and Binary Operators: Authors do write the codes containing mathematical manipulations. Also, for some conditional manipulations they need logical operators. So, they need arithmetic or logical operators either unary or binary. It's the count of unary and binary operators we have taken into consideration.

16. No. of Loops: This metric captures how frequently the author uses the loop. It's the no. of loops overall i.e. while, do-while, for etc an author have used collectively.

17. Single Literal Variables: This metric we hope can make the difference. It considers the single letter variables such as int a, int b etc. Because, reducing the no. of variables is the sign of efficient programming. It's the count we have taken of total no. of single literal variables.

18. Double Literal Variables: This metric considers the double literal variables such as int aa, int bb, int cc etc. This is also a counting type of metric.

19. Naive Variables: The concept of naive variables consist of the variables of the form int a1, int a2, int a1234 etc. i.e. those variables whose first character is alphabet and all other characters are digit. This is a Boolean type of metric.

20. Method Chaining: This metric deal with no. of method chaining done by an author. Method chaining consist of chaining of different methods one after the other. E.g.: Integer.equals(int i).toString (). Here equals (int i) and toString () are the two methods that are chained.

3.1.1 Boolean Metrics:

We are going to introduce another type of metrics called as Boolean metrics. In, counting type of metrics the actual count of the metric is taken into consideration. In Boolean metric only the presence or absence of that metric is taken into consideration. Following table shows the counting metrics(see Table 2) and Boolean metrics (see Table 3) from the above enlisted metrics.

Table 2: Counting Metrics

Counting Metrics	
Leading spaces	Average procedure length
Trailing spaces	Average indentations
Leading tabs	No. of methods
Trailing tabs	Unary and Binary operators
Line length	No. of loops
Lines of code(LOC)	Single literal variables
Brace position	Double literal variables
Comments	

Table 3: Boolean Metrics

Boolean metrics	
i-as-iterator	Naïve variable names
Conditional operator	Method chaining
Try statements	

3.2 Proposed System Architecture:

Here, the procedure begins with taking the source codes of the known authors as the input and give them to the feature extraction model for filtering purpose. Filtering includes scanning the inputted codes and extracting above source code metrics from them by using any java parser. As mentioned above filtering also take care of the values of these extracted metrics and gives them to model for classification. The decision tree based classifier such as ID3 or C4.5 can be used for classification. The classifier then based on values of extracted metrics takes the decisions and forms the classification model of known authors. This classification model is saved for future use. Then we take the input as source codes of unknown authors, again do the process of metric extraction [i.e. filtering] with those codes and obtain their values. Then classifier do the classification of codes of unknown authors based on above classification model we have built. (See Figure 1)

3.3 Proposed Algorithm:

1. Start
2. Take the source codes of known authors
3. Extract different metrics from those codes and do the training of those source codes
4. Generate classification model using decision tree algorithms.
5. Save classification model for future use.
6. Take the unknown codes to be tested.
7. Extract the metrics of that unknown codes
8. Classify the likely authors of the unknown codes using above classification model.
9. End

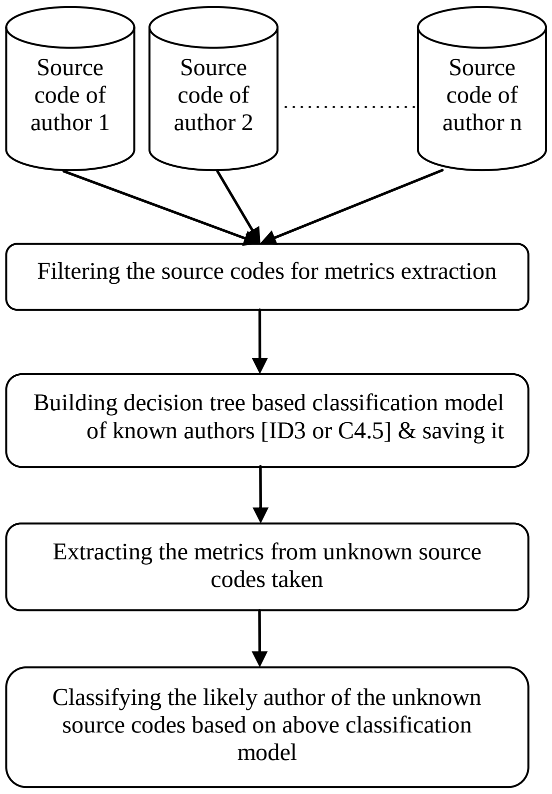
Source code of author 1
Source code of author 2
Source code of author n
Filtering the source codes for metrics extraction
Building decision tree based classification model of known authors [ID3 or C4.5] & saving it
Extracting the metrics from unknown source codes taken
Classifying the likely author of the unknown source codes based on above classification model

Fig 1: System Architecture

3.4 General Working of Decision Tree Based Algorithm:

Different people in the field of author identification uses different techniques as mentioned above. The proposed technique for author identification includes decision tree algorithms such as ID3, C4.5 etc [6]. The proposed method includes the selection of splitting attribute that splits the entire dataset into smaller subsets. Any attribute having the highest information gain amongst all can be the splitting attribute. Information gain can be calculated as the difference between the entropy of the original dataset and the weighted sum of entropies from each of the subdivided datasets. This procedure of selection of splitting attributes with the highest information gain and making subdivision of datasets continues recursively until all elements in each final subset belongs to the same class. Here, splitting attributes are nothing but metrics extracted. These algorithms calculate the information gain for each extracted metric and then choose the metric that has the highest information gain. Based on that metric, dataset is divided into subsets and this procedure continues until the entire classification has been done. While making the subdivision of the datasets a question of YES/NO type is asked to that metric i.e. splitting attribute and then the decision is taken for the division. Then, that metric is chosen as the tree node and entire tree is built by repeating the above procedure. While building a tree especially in C4.5 the concept called as pruning a tree exist. The pruning concept deals with the reduction of size of the tree by avoiding the sub trees or some branches of the trees which are not important and replacing them directly with the leaf nodes of theirs providing an equivalent representation.

3.5 General Decision Tree Based Classification Algorithm [ID3, C4.5 etc.]:

1. Start
2. Calculate entropy of entire dataset provided
3. Calculate entropy and weighted some of each of the metric
4. Calculate the information gain on each metric
5. Select the metric with the highest information gain as the splitting attribute
6. Take the decision of splitting the entire dataset available into subsets on the value of that metric selected
7. Make splitting attribute metric as the node of the tree.
8. Repeat
   Step 2 to step 7 until entire classification has been done or no instances remained for classification.

4. APPLICATIONS

Up till now we have seen the software forensic field, its applications, different software source code metrics, different techniques used to extract those metrics. Now, this section discusses about the applications of author identification. Those are as follows:

4.1 Plagiarism Detection:

Paul Clough [7] has discussed the term plagiarism and current technologies for plagiarism detection. The phenomenon of plagiarism includes the use of original work of some other person without taking the prior permission of that person or without acknowledging him. This may be intentional or unintentional sometimes. In most of the cases it happens in the areas of software education, where student copy each other’s code to fulfill the task. So, author identification is the one method due to which we can find the suspected plagiarism if the codes are exactly copied. Because, for exactly copied code the author’s style remains same as far as original author is concerned and the other claiming author can be identified easily.

4.2 Masquerade Detection:

Boleslaw K. Szymanski , Yongqiang Zhang[8] discussed the recursive data mining concept for masquerade detection and author identification. Author identification can also be very useful for masquerade detection. We know that viruses such as Trojan horse etc. are too dangerous for the system. So, it is very much essential to identify such viruses and find the solutions to them as early as possible. If a new piece of code causing the danger to system is identified, then there may be a possibility that it could be a virus. So, we can analyze the source code and try to find the author of that virus source code. If we find the author of the similar kind in our records, then we will try to neutralize that virus with the help of solutions we have previously done with this type of author.

4.3 Copyright Disputes:

Another application of author identification is resolving copyright disputes. Copyright means a person or an organization has the sole right of a particular work carried out by them. Without their permissions one cannot reproduce the work. So, in some cases the copyrighted work has been used without the permission of the author or reproduced it intentionally without the owner's knowledge. In this case, situations can be resolved by author identification, by actually identifying the likely author of the original work.

4.4 Software Maintainability:

As far as software organizations are considered, the hectic job for them is to maintain software in its good condition throughout its journey. Suppose for e.g., if a software is developed by a team of some people module wise. So, if in future, a problem occurs to a particular module of the software, the placing of that person who has developed that module in the past is necessary to provide an immediate service. In, that case, author identification can be used for finding that author from a huge pool of employees.

5. CONCLUSION:

This paper discusses about the general term software forensic and its applications. Also, the main perspective in this paper is author identification. Author identification is the phenomenon of the likely identification of the author of a given piece of a source code. This can be done by the process of metric extraction from, the different source code of the known authors and also from the piece of the source code to be identified. Then it is followed by the building of classification model for the known authors and then classifying the unknown authors based on that classification model. The paper also discusses about types of metrics and different metrics that are considered for proposed work. Then, the discussion is followed by different techniques of metrics extraction as well as author identification such as genetic algorithm, n-gram profiles, and neural network based algorithms etc and then moved towards proposed technology of decision tree based algorithm. Then it is followed with the discussion of different applications of author identification. Hope this paper helps you to gain the basic knowledge of the field of software forensic and its special application called "Author Identification"

6. REFERENCES:

[1] MacDonell S.G., Buckingham D., Gray A. R., and Sallis P. J. (2002) , Software Forensics : Extending Authorship Analysis Techniques to computer programs , Journal of Law and Information Scienece, 13(1) , pp. 34-69

[2] Jay Kothari, Maxim Shevertalov, Edward Stehle, and Spiros Mancoridis .A probabilistic approach to source code authorship identification", 4th International Conference on Information technology, IEEE Conference Publication, 2007.

[3] R. A. Vivanco, N. J. Pizzi, Identifying Effective Software Metrics Using Genetic Algorithm , Canadian Conference on Electrical and Computer Engineering, 2003, IEEE CCECE 2003.

[4] Georgia Frantzeskou, Efstathios Stamatatos, Stefanos Gritzalis, Sokratis Katsikas ,Source Code Author Identification Based on N-gram Author Profiles , Artificial Intelligence …, 2006 – Springer

[5] Frantzeskou G, Gritzalis S., & MacDonell S., (2004) ,Source Code Authorship Analysis For Supporting the Cybercrime Investigation Process , 1st International Conference on E-Business and Telecommunication networks. Setubal, Portugal, INSTICC Press, pp. 85-92.

[6] Margaret H. Dunham, Data Mining, Introductory and Advanced Topics, 4th Edition.

[7] Paul Clough – July 2000, Plagiarism in natural and programming languages: an overview of current tools and technologies.

[8] Boleslaw K. Szymanski,, Yongqiang Zhang , Recursive Data Mining for Masquerade Detection And Author Identification, Proc. 5th IEEE System, Man and Cybernetics Information Assurance Workshop, West Point, ,NY,June.2004,pp. 424-431