CROSS-ENTROPY APPROACHES TO SOFTWARE FORENSICS: SOURCE CODE AUTHORSHIP IDENTIFICATION

By

James Thomas Stinson

A Dissertation
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy
in Computer Science
in the Department of Computer Science and Engineering

Mississippi State, Mississippi

December 2011

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

CROSS-ENTROPY APPROACHES TO SOFTWARE FORENSICS: SOURCE CODE AUTHORSHIP IDENTIFICATION

By

James Thomas Stinson

Approved:

David A. Dampier Associate Professor of Computer Science and Engineering (Major Professor and Director of Dissertation)	Rayford Vaughn William L. Giles Distinguished Professor Associate Vice President for Research (Committee Member)
T. J. Jankun-Kelly Associate Professor of Computer Science and Engineering (Committee Member)	Cary Butler Technical Director, Information Technology Laboratory, U.S. Army Engineer Research and Development Center (Committee Member)
Edward B. Allen Associate Professor of Computer Science and Engineering (Graduate Coordinator)	Sarah A. Rajala Dean of the Bagley College of Engineering

Name: James Thomas Stinson

Date of Degree: December 9, 2011

Institution: Mississippi State University

Major Field: Computer Science

Major Professor: David A. Dampier

Title of Study: CROSS-ENTROPY APPROACHES TO SOFTWARE FORENSICS: SOURCE CODE AUTHORSHIP IDENTIFICATION

Pages in Study: 178

Candidate for Degree of Doctor of Philosophy

Identification of source code authorship can be a useful tool in the areas of security and forensic investigation by helping to create corroborating evidence that may send a suspected cyber terrorist, hacker, or malicious code writer to jail. When applied to academia, it can also prove a useful tool for professors who suspect students of academic dishonesty, plagiarism, or modification of source code related to programming assignments.

The purpose of this dissertation is to determine whether or not cross-entropy approaches to source code authorship analysis will succeed in predicting the correct author of a given piece of source code. If so, this work will try to identify factors that affect the accuracy of the algorithm, how programmer experience determines accuracy, and whether a cross-entropy approach performs better than some known source code authorship approaches. The approach taken in the research effort will manufacture a corpus of source code writings from various authors based on the same system descriptions and varying system descriptions, from which benchmarks of different approaches can be measured.

DEDICATION

I dedicate this work to my Father, the hardest working man I have ever known, who taught me the value of hard work, dedication, and most of all sacrafice. He sacrificed more for me than I will ever know. Without his examples and guidance, I would never have made it to point where this oppurtunity would have presented itself. I would also like to dedicate this work to my Grandmother, who taught me the value of education. A retired school teacher, she instilled the values in me to appreaciate learning, and to understand its importance to society. She lit a fire for knowledge in my mind that burns to this day. For this, I thank her.

ACKNOWLEDGEMENTS

The author expresses sincere gratitude to the following individuals and groups for providing support throughout this research work:

- My wife, without whose support I could not do anything in this world,
- The author's PhD advisor, academic mentor, and constant source of moral support and solace,
- The author's PhD committee and all of MSU's CSE instructors who contributed to the author's-education,
- The instructors and professors at MSU who took time out of their busy schedules to help gather the data set necessary to complete this work,
- Supervisors and colleagues at the U.S. Army Engineer Research and Development Center, and
- Countless friends, acquaintances, and mentors.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

The threat of malicious code affects most computer users. Millions of dollars in productivity a year are lost by businesses; cherished moments in life captured by individuals are lost; important documents are destroyed; corporate, government, and personal information is compromised; money is wasted on repairs and protection; and life is interrupted. Malicious code is a problem, so much so that an entire industry is predicated upon providing a solution. McAfee™, Symantec™ and AVG™ rake in millions in revenue to help protect users from unwanted attacks. Huge databases are assembled comprising all known malicious code signatures in an effort to protect unknowing users from the dangers that lurk on our networks.

The anonymity of the Internet provides a safe haven for malicious code writers. Their methods are ingenious and their results are often devastating. Who's responsible, that is the question. As honest people, how do we bring these havoc wreakers to justice? How do we use science to help prove their complicity? With confidence, how can we show that a particular person, or group of persons, is the responsible party, in this case the author of a particular malicious code?

Determining the author of a given computer program is difficult. Computer source code is rigid and structured; however, there is enough flexibility within the framework that individual characteristics are transferred to text, somewhat like a fingerprint. The purpose of this work is to determine whether or not a cross-entropy

approach to source code authorship analysis will succeed in predicting the correct author of a given source code. A cross-entropy approach to author identification has been applied to literary works of multiple natural languages; however it has not been applied to authorship attribution for source code. This work will focus on factors that affect the accuracy of the cross-entropy approach, how programmer experience determines identification accuracy, and whether or not a cross-entropy approach performs better or worse than some other known source code authorship approaches.

The remainder of this chapter is organized as follows: Sections 1.1 and 1.2 summarize the scope of digital forensics and software forensics, Section 1.3 discusses the aspects of software forensics, while Section 1.4 discusses the differences between authorship identification and plagiarism detection. Section 1.5 presents the motivation and applications for this work, while Section 1.6 discusses the hypotheses and research questions and highlights the contributions of this dissertation. Section 1.7 provides an overview of the remainder of this document.

1.1 Digital Forensics

In an attempt to better deal with the problems associated with malicious attacks and other computer-related crimes, the First Annual Digital Forensic Research Workshop was held in 2001 where the scope of Digital Forensics was formally defined: "The use of scientifically derived and proven methods toward the preservation, collection, validation, identification, analysis, interpretation, documentation, and presentation of digital evidence derived from digital sources for the purposes of facilitating or furthering the reconstruction of events found to be criminal, or helping to anticipate unauthorized actions shown to be disruptive to planned operations"[1].

Of the three major categories identified, code analysis (what has become known as software forensics) was seen as an important part of the investigative process. Borrowing extensively from the existing fields of linguistics, natural language processing, artificial intelligence, and software metrics, software forensics has become a burgeoning research area.

1.2 Software Forensics

Computer programs are generally written in a quasi-natural language, what is known as a programming language; at its most basic form it is referred to as source code. Source code differs from natural language texts, such as stories, emails, memorandums, etc., because it is highly structured. In addition, source code contains a number of keywords, which are repeated in patterns throughout the document. Anyone who has taken a programming language class understands the nature of source code. A language is flexible, yet must adhere to strict standards. Lines must be terminated, keyword declarations included before variables, parentheses used to signify functionality, etc. It must do so in order for the machine (a computer) to understand how to interpret the programmer's instruction. In practice, a programming language is essentially a protocol for communication between a person and a computer, which is explicitly defined and constrained in order to clarify the programmer's intentions.

For example, when defining an operation in source code, does the programmer wish for the functionality to be public, what value will be returned, and how should it be typed so it can be interpreted, what parameters or inputs will be needed and how should they be encoded, and so on and so forth. Anyone who has written a parser for a compiler or interpreter understands why. The structure helps tell the computer specifically how the

program should operate and what actions are allowable. Machines have no intuition or background context; therefore everything provided for them must be specified and deliberate.

Languages for writing computer programs differ in terms of their so-called generation (roughly the time that they were devised and reflecting their level of abstraction) and type (such as procedural, declarative, object-oriented, and function) [2]. However, while it would be extremely difficult to draw comparisons for authorship attribution across languages, programs written in the same or very similar languages can be examined from a forensics viewpoint.

Take, for example, the code fragments written in the popular language C++ from [2], shown in Figure 1.1. Both programs provide the same functionality (calculating the mathematical function factorial(n), normally written as n!). While the source code looks very different, each snippet of code produces the same output. What is captured in these source code snippets are the different styles of the authors. A closer examination reveals that one author used recursion (usually considered a more eloquent solution applied by experienced programmers) while the other used a for loop for repetition. Also, the first snippet would appear to be more human-readable. Each variable is defined on a separate line; comments are provided; variable names are more meaningful; and declarations, terminations of assignments, or executions are separated into individual lines.

From the snippets shown in Figure 1.l, it becomes obvious that source code is certainly more restrictive than natural language; however, source code authors still have a large degree of flexibility when writing a program to implement an idea or functionality. This flexibility can be used to identify the writing characteristics of a particular author.

Examples of flexibility can include the way source code is physically structured in terms of layout, such as space, indentation, bracket placements, etc.

```
//Factorial takes an integer as input and returns the factorial
//Code snippet 1

int Factorial (int number)
{
    //declare local variables
    int factorial;
    int counter;
    fact=1;  //initiate to 1 since factorial 0 is 1
    for (counter=number; counter>1; counter--)
    {
        factorial=factorial*counter;
    }
    return factorial;
}
```

```
//Factorial takes an integer as input and returns the factorial
//Code snippet 2

int fact(int x){
if (!x) return 1; else return x*fact(x-1);}
```

Figure 1.1 C++ code snippets show the same functionality [2].

Beyond layout characteristics, stylistic choices can also include the use of variable names, preferences for control structures or program statements over others (e.g., *for* loop versus *while* loop), levels of functional decomposition, the use of properties versus accessor methods, or accessor methods versus global variables, etc. In addition, programmers have freedom to choose computer platform, language, compilers, and editors, although a description of these freedoms will go beyond the scope of this work.

These features can be quite specific to certain programmers or types of programmers [2]. When combined to form particular combinations of features and

programming idioms, they can become telling signs related to a programmer's problem-solving vocabulary. Therefore, it seems that computer programs can contain some degree of information that provides evidence of the author's identity and characteristics [3].

1.3 Aspects of Software Forensics

Software forensics is an investigation of source code characteristics. According to Sallis, Aakjaer, and MacDonell [3] and Gray, MacDonell, and Sallis [2], four principal aspects of authorship analysis (outlined below) can be applied to software code (Figure 1.2). Their definitions are as follows:

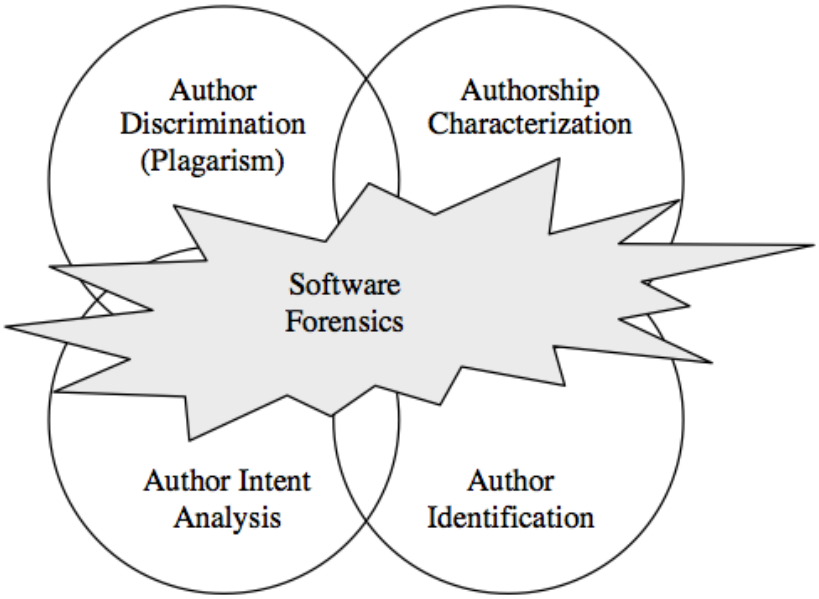

Figure 1.2 Software Forensics [2].

1. Author discrimination. Deciding whether pieces of code were written by a single author or by multiple authors, for example, showing that two pieces of code were written by different authors, without actually identifying the authors in question. This can also describe situations involving plagiarism.

2. Author identification or attribution. Determining the likelihood of a particular author having written some pieces(s) of code, usually based on code samples written by a programmer and available for analysis. For example, assigning authorship of a new piece of code, such as a computer virus, to an author where the code "matches" the profile of other code written by the author.
3. Author characterization. Determining some characteristics of the programmer based on source code analysis, for example, determining if a programmer is a novice or if he/she uses more eloquent methods for programming.. Characteristics may include personality traits, educational background, attention to details, etc.
4. Author intent determination. Determining whether code was deliberately malicious, or was the result of accidental errors or bugs. Developing software is rarely ever error free. when errors have catastrophic consequences, software forensics tries to determine whether the cause was due to negligence or was caused by malicious intent.

Software forensics investigates author identification or authorship attribution with specific focus on determining how to identify the correct "fingerprints" or "markers" that yield the best results when determining authorship. These markers can be based on stylistic metrics inherent in source code structure and author preferences, or can be sequences of n-grams, prefixes, mean match lengths, word grams, etc.

1.4 Authorship Analysis versus Plagiarism Detection

It may seem that authorship analysis and plagiarism detection are the same endeavor. Krsul and Spafford [4] argue that while they are most certainly related, there

are some significant differences between the two. Plagiarism can be defined as the complete, partial, or modified replication of software, with or without the permission of the original author [4]. By this definition, plagiarism detection cannot discern whether the same individual wrote two separate programs. The key point is that the replication of code does not necessarily entail keeping the programming style of the original piece of code. Stylistic changes can be made to the source code while the functionality remains the same, blurring the lines of authorship. While plagiarism detection needs to discover the similarity between these two programs, authorship analysis does not [4]. The functionality of a code may be stolen or copied by a plagiarist; however during the copying process the code itself may undergo several stylistic changes that mask the identity of the original programmer. A plagiarism detection system might consider the two identical, where an authorship analysis system may not [4].

1.5 Motivation and Application

Some of the earliest works published on source code authorship identification date back to the late 1980's or early 1990's [5-7]. New to the computing scene were malicious codes that threatened the evolving internet and emerging PC markets. Viruses, worms, Trojan horses, and crackers began to leave systems compromised with owners and system administrators feeling victimized. Often, system users would be unaware that attacks had taken place until well after the event had occurred. The task then became one of reconstructing the crime scene, determining what happened, how the system was compromised, and who was responsible for the intrusion. From these ideas, researchers began to wonder whether or not it would be possible to identify the author of a piece of source code from the small snippets left behind.

The ultimate goal of source code authorship identification is to create techniques, or combinations of techniques, that can be applied in a legal setting to assist courts in making judgments. While expert opinion can be given based on degrees of similarity and differences between source code fragments, a more scientific approach is desired whereby both qualitative and quantitative metrics are used to show correlations between code fragments formally. Additionally, scientific approaches lend themselves to processes and automation, thereby creating procedures and toolkits that help reduce human errors and aid in the investigation and analysis processes.

Krsul and Spafford [4] have identified four basic areas that can directly benefit from the development of authorship analysis techniques:

1. In the legal community, methodologies are needed that can be used to provide empirical evidence to resolve authorship disputes.
2. In the academic community, it is considered unethical to copy programming assignments. While plagiarism detection can show that two programs are equivalent, authorship analysis can be used to show that some code fragment was indeed written by the person who claims authorship of it.
3. In industry, where there are large software products that typically continue for years, and contain millions of lines of code, it is a common occurrence that authorship information about programs or program fragments is nonexistent, inaccurate, or misleading. Whenever a particular program module or program needs to be rewritten, the author may need to be located. It would be convenient to be able to determine the identity of the programmer who wrote a particular piece of code from a set of several hundred programmers so he can be located to assist in the upgrade process.

4. Real-time misuse detection systems could be enhanced by inclusion of authorship information. A programmer signature constructed from the identifying characteristics of programs constitutes a pattern that can be used in the monitoring of abnormal system usage.

1.6 Hypothesis and Research Questions

1.6.1 Hypothesis

This dissertation is an attempt to extend Patrick Juola's work on authorship identification for literary works into the domain of computer software forensics. Juola has successfully used a cross-entropy technique to identify language relationships, document topic and genre, and authorship attribution for various literary works [8-10]. The hypothesis for this dissertation is defined as follows:

The cross-entropy approach, which has been applied successfully to literary documents, will be at least as accurate as other known approaches to the identification of program source code authorship.

To understand cross-entropy, one must first understand entropy. Entropy is an attempt to measure uncertainty based on probability. It is defined in information theory by Shannon as the unpredictability of a given event, provided that all relevant information is brought to bear [10-12]. In other words, given a sequence of messages, what is the probability that the next message will be X? Cross-entropy is a measure of the unpredictability of a given event, given a specific (but not necessarily best) model of events and expectations [10] (in this case, candidate source code samples). Given a model, in this case a piece of source code of known authorship where the source code is

the message source, what is the measure of entropy between the model and the test or candidate document?

Juola's cross-entropy technique is based on previous work by Wyner [13][14]. Wyner's estimation technique has been shown to be an efficient method for entropy estimation, especially when a limited sample size is available. In fact, this is the strength of Wyner's approach, which can prove to be a valuable approach in software forensic investigations where source code samples are not readily available for law enforcement. It was developed by Wyner et al. and is outlined in [13] and [14]. The strength of the algorithm lies in its sliding window approach, providing many sample "messages," which gives a better measurement of probability distribution. To measure this distance a sliding window of size *n* is applied to the source code text. The information inside the sliding window becomes the database (or profile) against which a test source code document, of unknown authorship, is compared. The goal is to find the longest continuous prefix that matches what is in the current database. The longest prefixes are then averaged over the sliding window. The longer the length of the match, the more similar the documents are. A more detailed description of cross-entropy will be discussed in Chapters II and III, starting with section 2.4.3.

1.6.2 Research Questions

The literature reviews for this dissertation will be within the scope of the following topics: software forensics, language processing techniques, clustering and distance algorithms, and other artificial intelligence literature that is relevant to document and source code authorship attribution and analysis. Those methods that are already established for performing source code authorship identification will be discussed in

Chapter II. Evaluation of the hypothesis is based on the research questions in the following paragraphs. More detail of the experimental design and additional research questions can be found in the latter sections of Chapter III. The main research question investigated in this work is as follows:

Can a cross-entropy approach be used to predict source code authorship? Cross-entropy has been shown to identify authors in literary works. Will cross-entropy approaches taken in literary document classification and authorship identification fare similarly when compared with cross-entropy approaches applied to source code authorship identification?

First, this problem will be explored by introducing an implementation of the cross-entropy algorithm, outlined in [8-10], to multiple source code corpora written by professional and student authors. The cross-entropy algorithm will then be applied to all corpora, and the results will be tabulated.

Is cross-entropy more or less accurate than other approaches when determining source code authorship, such as, N gram based approached outlined in Chapter II.

This problem will be explored by applying a state-of-the-art nonmetric-based approach (an N-Gram approach) to these same corpora to gauge the effectiveness of cross-entropy as a viable option. This is important because cross-entropy, to the author's knowledge, has not been applied to source code analysis. A measuring stick is needed to compare results and perform meaningful analysis.

Also, the assumption is made that experience should help the algorithm discern authors because distinctive traits are developed with experience, as older habits are tried, true, and reliable. Therefore, the one research hypothesis to be tested is as follows:

The cross-entropy approach will more accurately identify experienced programmers when compared against less experienced programmers.

Obviously, the professional corpora will comprise experienced programmers. To form the student corpora, accuracy scores from assignments later in the semester will be examined, assuming students have gained experience over the course of the semester. In addition, most student code files, per assignment, will be roughly the same file size/lines of code and will probably be much smaller than submitted professional code files. Therefore, a comparison between the professional corpora and the student corpora will be investigated in order to answer the following research question:

How does the size of the author's profiles affect performance of the approach? Will a larger profile, or larger file size, say 2000 lines of code per author, cloud the results of the algorithm or help fine-tune towards a more discernable identification, or have no difference?

As mentioned previously, cross-entropy is a measure of the unpredictability of a given event, given a specific (but not necessarily best) model of events and expectations [10]. More source code would seem to give a better representation of the model. It would seem the more code available to the method, the more opportunities for distinctive identifiers to manifest themselves in the corpora; therefore, one testable hypothesis is:

More lines of code per author should yield better predictive results.

As briefly mentioned above, a major difference between the professional corpora and the corpora of students is the purpose of the code samples. The source code corpora composed of professional code differs greatly from the student corpora because the professional code samples will not be a duplication of the same functionality, i.e., a programming assignment, but rather contributions of code pieces in a team environment

with a common goal. The computer science student corpora will have the same objective with regard to functionality. This leads to the next research question, which is as follows:

Will the cross-entropy approach more accurately identify authors where the test-bed programs have the same functionality/objective, versus programs that have varying objectives? Will the algorithm be able to detect functional similarity?

In other words, will this reduction in noise help the algorithm differentiate between programmers; or because more authors are using the same constructs, will it restrict the performance of the algorithm because source code samples are now more similar, blurring the lines of differentiation? This will be especially true considering the nature of the student corpora.

Beyond functional/objective similarity of code samples, the cross-entropy approach uses a sliding window of size n. *To further investigate accurate classification with respect to window size, the performance of the cross-entropy algorithm will be explored and documented in an effort to understand the optimal window size for various code types.*

What is the role of window size with respect to correct classification with respect to functional objective, experience, size, etc.?

Windows where *n* is small may not capture author identification fingerprints completely or give high marks for matches that are a result of language constraints. Window size will be investigated to determine relationships with respect to accuracy. A more detailed discussion of research questions is addressed in Chapter III.

1.6.3 Contributions

Below is a list of expected contributions:

1. Determination of whether or not cross-entropy can be used as method for determining authorship of source code.
2. Results detailing whether the method is more or less accurate than other known methods.
3. A corpus of source code documents that can be used in this research and future research activities.
4. Results detailing how source code sizes, programmer experience, functional objectives, etc., affect cross-entropy results.

1.7 Publication Plan

Table 1.1 presents a list of upcoming forensic conferences where the research showcased in this work may be presented.

Table 1.1 List of Conferences

DoD Cyber Crime Conference 2012	Jan 20-27 Atlanta, GA	http://www.dodcybercrime.com/12CC/index.asp
2012 American Academy of Forensic Sciences Annual Meeting	Feb 20-25 Atlanta, GA	http://www.aafs.org/aafs-2012-annual-meeting
12th Annual CanSecWest Conference	Mar 09-11 Vancouver, British Columbia, Canada	http://cansecwest.com/
Computer Enterprise and Investigation Conference	May 21-24 Summerlin, NV	http://www.ceicconference.com/
Techno Security 2012	Jun 03-06 Myrtle Beach, SC	http://www.techsec.com/html/Security%20Conference%202012.html
Mobile Forensics Conference	Jun 03-06 Myrtle Beach, SC	http://www.mobileforensicsconference.com/

1.8 Organization

The remainder of this dissertation provides the background and details of the research work. Chapter II reviews related work in the areas of software forensics, artificial intelligence, and natural language processing. Chapter III describes the experimental design, methods, and additional research questions for the cross-entropy approach. Chapter IV presents the results and comparisons of the experiments. Finally, Chapter V presents conclusions and identifies potential areas for future research.

CHAPTER II

RELATED WORK

2.1 History of Authorship Attribution

Authorship attribution, the process of determining the author of a particular text, is a field that is over 120 years old. The pioneering works of Mendenhall trace back to 1887 [15] with work focusing on Shakespearian plays, while in the early twentieth century Yule [16, 17] and Zipf [18] emphasized statistical methods and relative frequencies of word distributions. However, the seminal work for modern authorship attribution studies began with Mosteller and Wallace [19], whose work in 1964 focused on a corpus comprising *The Federalist Papers* (146 political essays written by Alexander Hamilton, James Madison, and John Jay, with 12 being claimed by both Madison and Hamilton). This study made the cover of *Time* magazine and garnered the attention of academics and the public alike.

This work is considered important because it took focus away from traditional human expert-based methods and shifted it toward more scientifically quantifiable techniques. Until the end of the twentieth century, most research focused on "stylometry," or trying to identify features within a text that are inherently tied to a person's writing style [20, 21]. Researchers developed various measures, or metrics-based approaches, to help identify style.

This chapter will examine in detail various approaches and results, both stylometric and nonstylometric based, related to software forensics, specifically

authorship analysis identification. The chapter will also discuss the differences between metric-based approaches and a discussion of the cross-entropy approach and the inner workings of the algorithm when applied to literary works.

2.2 Metrics-Based Approaches to Software Forensics

Most of the source code attribution work from the early 1990's to the mid 2000's focused heavily on the identification and use of metrics, sometimes referred to as markers, to help determine authorship. These "traditional" metric-based approaches [2, 3, 5-7, 22-24] follow a paradigm composed of two steps. In the first step software metrics are identified and extracted in an attempt to create and capture a "fingerprint" or "DNA" that represents the author's style. The second step uses the identified metrics to develop models that are capable of discriminating between different authors using various classification algorithms.

2.2.1 Traditional Metrics-Based Approaches

The term software metric was defined by Conte, Dunsmore and Shen [25] as follows: "Software metrics are used to characterize the essential features for software quantitatively, so that classification, comparison, and mathematical analysis can be applied." These metrics, usually a researcher-defined list of "markers" to quantify, are what metrics-based techniques try to analyze when performing authorship analysis.

In 1989 Oman and Cook [26] proposed the idea of identifying authorship by developing methods for "fingerprinting" programs to determine instances of software theft and plagiarism. Their methods consisted of two steps: (1) the comparison of the structural decomposition of the systems or program under investigation, and (2) the application of a battery of software complexity metrics to identify suspect programs.

They termed these metrics as marker characteristics, which represent the unique features of a programmer's style. The markers were most closely associated with typographic or layout characteristics, such as line length, comment formats, spacing and indentations, the use of blank lines, etc. However, the significance of their work is that they showed that complexity markers were not good indicators of authorship.

Complexity metrics are not the same as the typographic or stylometric markers listed above. Complexity metrics focus on the complexity of the functions within the source code, such as the number of lines of code, number of lines of comments, number of arguments, operator counts, cyclomatic complexity, level of nesting, etc. Most plagiarism detection of the early 1980's focused on complexity analysis [27, 28]. In addition, their fingerprint markers were generally invariant with respect to the problem requirements.

Oman and Cook's experiment used Pascal source code containing three algorithms from six different computer science textbooks. The algorithms contained segments for bubble sort, quicksort, and a set of tree traversal algorithms. The code samples were then given to human subjects, who were tasked with grouping the code based on who they thought the author was. Most of the subjects correctly grouped the author based on analyzing style and structure. From the discussions with the subjects, a finite set of metrics was identified from which an automated analyzer was constructed. Specifically, the analyzer generated a Boolean value for each of the following conditions [26]:

1. Inline comments on the same line as source code.
2. Blocked comments (two or more comments occurring together).
3. Bordered comments (set off by repetitive characters).

4. Keywords followed by comments.
5. 1- or 2-space indentation most frequently occurring.
6. 3- or 4-space indentation most frequently occurring.
7. 5-space or greater indentation most frequently occurring.
8. Lower case characters only (all source code).
9. Upper case characters only (all source code).
10. Case used to distinguish between keywords and identifiers.
11. Underscore used in identifiers.
12. BEGIN followed by a statement on the same line.
13. THEN followed by a statement on the same line.
14. Multiple statements per line.
15. Blank lines in the declaration area.
16. Blank lines in the program body.

According to Oman and Cook, the above measures, or markers, were highly accurate across modules written by an author with a consistent style. Inconsistent authors were harder to measure. Based on the Boolean results, a clustering analysis using the Hamming distance was applied with the purpose of grouping code segments with authors. Figure 2.1 [26] shows the results of the analyzer's scoring while Figure 2.2 [26] shows a distance matrix of the cluster analysis.

Oman and Cook's metrics-based approach would dominate source code authorship attribution research throughout the 1990's. Subsequent efforts would expand upon stylometric research with decent results.

Typographic Style Vectors for Textbook Data

	INL	BLK	BOR	KEY	I2	I4	I5	LCO	UCO	<C>	U_S	BGN	THN	;;;	BLD	BLB
Text A (IR = 0)																
Bubblesort	1	1	0	1	1	0	0	0	1	0	0	0	0	0	1	1
Quicksort	1	1	0	1	1	0	0	0	1	0	0	0	0	0	1	1
Tree Traversal	1	1	0	1	1	0	0	0	1	0	0	0	0	0	1	1
Text B (IR = 1)																
Bubblesort	1	1	1	1	1	0	0	0	0	1	0	0	1	0	0	0
Quicksort	1	1	1	1	1	0	0	0	0	1	0	0	1	0	0	1
Tree Traversal	1	1	1	1	1	0	0	0	0	1	0	0	1	0	0	0
Text C (IR = 2)																
Bubblesort	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0
Quicksort	0	0	0	0	0	1	0	0	0	1	0	0	0	0	1	1
Tree Traversal	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0
Text D (IR = 3)																
Bubblesort	1	1	0	1	0	0	1	1	0	0	0	0	1	0	0	0
Quicksort	1	0	0	1	0	0	1	1	0	0	0	0	1	0	1	1
Tree Traversal	1	0	0	1	0	0	1	1	0	0	0	0	1	0	0	0
Text E (IR = 6)																
Bubblesort	1	0	0	0	1	0	0	1	0	0	0	0	1	1	1	1
Quicksort	1	1	0	1	1	0	0	1	0	0	0	0	1	1	1	1
Tree Traversal	0	0	0	0	1	0	0	1	0	0	0	0	1	0	0	0
Text F (IR = 6)																
Bubblesort	1	0	0	1	0	0	1	1	0	0	0	1	0	1	0	0
Quicksort	1	0	0	1	0	1	0	1	0	0	0	1	0	1	0	0
Tree Traversal	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0

INL - inline comments
BLK - block of comments
BOR - bordered comments
KEY - comments after keywords
I2 - 1 & 2 space indentation
I4 - 3 & 4 space indentation
I5 - 5 & greater indentation
LCO - lower case only
UCO - upper case only
<C> - case distinguishes keyword
U_S - underscore used
BGN - Begin & statement on 1 line
THN - Then & statement on 1 line
;;; - many statements per line
BLD - blank lines in declarations
BLB - blank lines in the body
IR - Inconsistency Rating

Figure 2.1 Output of Fingerprint Analysis [26].

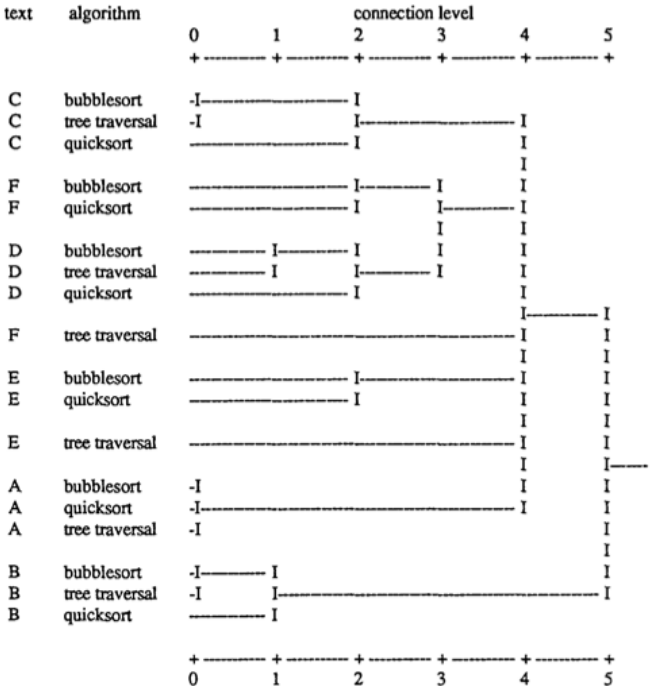

Figure 2.2 Clustering Analysis of Authors [26].

2.2.2 Software Forensics: Can We Track Code to Its Authors

With the emergence of the Internet in the early 1990's, malicious code became more prevalent, and plagiarism detection was joined by source code authorship attribution code as a larger field of study. In 1992, Spafford and Weeber [7] proposed a process called software forensics, where the features of malicious code remnants might be analyzed and then used to identify their authors. As a cornerstone of the software forensics process, Spafford and Weeber proposed a stylometrically based approach [7]:

> Programming, especially in a language rich in data types and control structures, has considerable room for variation and innovation. Even if coding is from detailed specifications, room exists for personalization. Programmers generally acknowledge that they each have a unique coding style. Using appropriate stylistic elements may help in the production, reuse, and debugging of code. Many texts recommend elements of style to use when programming, and often programmers integrate selected elements of others' styles into their own repertoire as they gain experience programming. The keys to identifying the author of suspect code are selection of an appropriate body of code and identification of appropriate features for comparison.

Among the features presented to reflect style, 11 are proposed that could be used to identify and author [7]:

1. Language: Programmers usually have a preferred language for development.
2. Formatting: The format exhibits a personal style and tends to be consistent between programs.
3. Special Features: The format contains pragmas or special macros that are not present on every system.
4. Comment Styles: Users tend to have a distinctive style for commenting on programs, including frequency and detail.

5. Variable Names: Some authors connect words with underscores, some capitalize, some use naming schemes.
6. Spelling and Grammar: Misspelled variables or comments are often misspelled consistently.
7. Use of Language Features: Authors prefer programming language features over other features.
8. Scoping: The ratio of global to local may be a trait.
9. Execution paths: Code is present that cannot be executed such as code that was present for debugging but not removed.
10. Bugs: Some authors consistently make the same mistakes in their code.
11. Metrics: Number of lines of code, comment to code ratio, function complexity, etc., are consistent.

While Spafford and Weeber propose metrics to take into consideration when analyzing source code, they provide no statistical evidence to support their theory. In 1996, Krsul and Spafford [24] wrote a lengthy, comprehensive report examining a myriad of different statistical methods applied to the metrics acquired from various C programming language codes. This section will take a comprehensive look at the approach experiments and results of this report.

Krsul and Spafford [24] present some common-sense challenge areas that should be taken into account when exploring the idea of software forensics. For example, the programming characteristics of programmers change and evolve over time. One huge factor is education. As a programmer advances, he or she will often become more proficient, adapt new techniques, and apply a greater deal of structure when programming to make code more eloquent and reusable. In addition, software engineering models that

aid developers by aiming to create maintainable, reusable code also impose such restrictions as naming conventions, commenting styles, and parameter passing methods. These measures force developers on a team to become more standard while reducing stylistic flexibility. Other factors identified are the use of formatting tools within Integrated Development Environments (IDEs) that auto place brackets, indentions, and comments. However, the most serious problem they identify is reuse. Copying and pasting other author's work is a common practice, especially in commercial development projects. Even today, programming communities and websites have sprung up where free samples of code are readily available for download for just about any problem imaginable. An individual would have little problem copying and pasting a few sample projects together to create a working application with most of the code coming from other sources. (However, an argument could be made that most malicious code is not shared at such sites. It is the author's opinion that malicious code is usually written in secrecy and is often ingenious in concept, with an individual or a small group of individuals responsible for its construction.)

The approach taken by Krsul and Spafford in their experiment is to identify a set of stylistic metrics, which in turn becomes the "feature set" that identifies the author's style. Note that feature sets can be anything that is used to capture information about the author. It does not have to be based on such metrics as indentation, bracket placement, variable names, or variable length; it can be based on other measures that capture frequencies or tabulations, such as n-grams, longest prefixes, etc.

1. Programming Layout Metrics: deal specifically with the layout of the program. Measurements for indentation, placements of comments, placement of brackets, etc., are taken into account. They are fragile because they can change

significantly based on editors and are ingrained by instructors when a student learns to program.

2. Programming Style Metrics: focus on statistical distribution of variable lengths, comment lengths, etc.
3. Programming Structure Metrics capture metrics that are dependent on programming experience and ability. Metrics include the number of lines of code per function, usage of data structures, etc.

The source code used in the experiment was derived from students, faculty, and staff at Purdue University, West Lafayette, IN. Once the metrics were gathered and organized, statistical analysis was performed. In the first phase of the experiment, discriminate analysis was performed: observations were first separated and new observations assigned to previously defined groups. Later, the experiment employed neural networks and likelihood classifiers in an attempt to discern authorship.

2.2.2.1 Discriminant Analysis

When initially applying discriminate analysis, Krsul and Spafford found poor results:

> This technique works best with those metrics that show little variation between programs (for a specific programmer) and large variations among programmers. Unfortunately, analysis of the metrics collected shows that these two criteria are not necessarily correlated. Initially, we calculated the standard error by programmer for every metric, and eliminated those that showed large variations because they identify those style characteristics where the programmer is inconsistent. Surprisingly, most the metrics that showed large variations among programmers were eliminated as well. The performance of our statistical analysis with the remaining metrics was discouraging, with only twenty percent of the programs being classified correctly. [24]

To resolve the issue, a tool was developed to measure programmer consistency. Two graphs were shown that displayed the variation of the metrics within programs for each programmer accompanied by the distribution of values for each metric among all programmers. An additional graph showed the consistency of each programmer for each metric. The idea is to show variations in values for a programmer without having a significant "jump" in consistency between consecutive programs.

Using this technique and choosing a small subset of the available metrics improved the success rate for classification to 73 percent. The algorithm routinely misclassified some students. Upon further review, it seemed that one student had radically changed his style over a period of 2 months while another seemed to have relied on class projects from previous semesters. However, the other misclassified programmers did show a consistent style, which led to two conclusions: (1) the metrics chosen for the experiment were not comprehensive enough to distinguish among authors, and (2) the statistical tool was not comprehensive enough [24]. For example, consider Figure 2.3, taken from [24]. Although the author of the code was consistent in coding style, he or she added comments at the end of the code blocks to indicate what block was ending. No metric was defined for comments at the end of code blocks.

```
main() {
.
.
.
  while ( !eof ) {
    ch = getch();
    if( ch == 0 ) {
          .
          .
          .
    } else {
          .
          .
          .
    } /* if( ch == 0 ) */
  } /* while ( !eof ) */
} /* main() */
```

Figure 2.3 An Undefined Metric, in This Case Comments at the End of Code Blocks,That Was Consistent within an Author's Style [24].

In addition, some programmers were consistently misclassified as other programmers, such as programmer 17, who was always misclassified as programmer 19. Krsul and Spafford hypothesize that this may have been a result of very similar styles. If the sets of code for the programmers were examined more carefully, it may have been possible to define a set of metrics to distinguish between the two. However, in a real-world application, where a court of law is involved, an approach such as this would not be feasible and may be considered biased.

2.2.2.2 Statistical and AI techniques

In the second phase of their experiment [24], Krsul and Spafford used LNKnet, software developed at the Lincoln Laboratory, Massachusetts Institute of Technology, Lexington, MA [29], to perform statistical analysis and classification. LNKnet "simplifies the application of some of the most important statistical, neural networks, and

machine learning pattern classifiers" [24]. Table 2.1, extracted from the LNKnet user manual [29], describes the statistical analysis and machine learning techniques available.

Table 2.1 LNKnet Algorithms [29]

	Supervised Training	**Combined Unsupervised-Supervised Training**	**Unsupervised Training (Clustering)**
Neural Network Algorithms	Back-Propagation (BP) Adaptive Stepsize BP Cross-Entropy BP Hyperspace Classifier Committee	Radial Basis Function (RBF) Incremental RBF (IRBF) Top-2-Diff IRBF Nearest-Cluster Classifier	Leader Clustering
Conventional Pattern Classification Algorithms	Gaussian Linear Discriminant Quadratic Gaussian K-Nearest Neighbor (KNN) Condensed KNN Binary Tree Parzen Window Histogram	Gaussian Mixture (GMIX) Classifier Diagonal/Full Covariance GMIX Diagonal/Full Covariance GMIX Tied/Per-Class Centers GMIX	K-Means Clustering E&M Clustering
Feature Selection Algorithms	Canonical Linear Discriminant Forward and Backward Search using N-fold Cross Validation		Principal Components Analysis

A caveat when using LNKnet for neural networks and classifications is that it requires large numbers of data points, in this case programs, to perform analysis. It needs a training data set, evaluation data set, and test data set. This may be possible in a corporate or academic setting, but in most cases the number of programs collected from malicious code writers during the investigation process will likely be modest at best.

Because of the small number of programs available for training and evaluation, Krsul and Spafford [24] applied N-fold cross-validation to provide the analysis algorithms with the necessary data points. The LNKnet User's Guide describes this as follows: "the idea of cross-validation is to split the data into N equal-sized folds and test each fold against a classifier trained on the data in the other folds." [29]

The LNKnet software [29] provides a feature search algorithm to determine the best-input features (identifying metric) in the data set. The feature selection search can go in three directions through data space: forward, backward and forward-backward.

1. Forward means each feature is tried alone and the feature with the best classification rate is selected as the first feature. The remaining features are tested in combination with this feature and the best combination is added as the second, as so on and so forth.
2. A backward search starts the search with all features selected and tries to leave each, meaning that the feature that the classifier did the best without is excluded, and so on until none are left. (The idea is some features do well together but poor individually.)
3. The forward-backward is a combination of the two methods. The search starts with no feature selected; when it adds two, it searches for one to take away. It continues adding two and taking away one, until it has exhausted all available features. This approach can find hidden dependencies between features.

While Krsul and Spafford [24] tried all of the classification methods provided by LNKnet, some did not perform well. They only listed only the five algorithms or approaches that yielded success rates above the 70 percent mark.

2.2.2.3 Neural Network

The Multi-layer Perceptron (MLP) neural network was able to classify at a high percentage. Error rates as low as 2 percent were achieved using Linear Discriminate Analysis (LDA) normalization, 4-fold cross-validation, 100 nodes in the hidden layer, cross-entropy as a cost function, and a forward-backwards feature search [24]. However,

Figure 2.4, taken from [24], shows how the addition of a few metrics dramatically decreases classification accuracy. In addition, the algorithm performed much worse when not using LDA as a normalization technique with the best error rate being 26 percent with 40 metrics [24].

2.2.2.4 Gaussian Classifier

Gaussian classifiers are simple classifiers that model each class with a distribution center around the mean of the class. The LNKnet manual [29] specifies that Gaussian classifiers "estimate a scaled probability density function or likelihood for each class, $p * (X|A) P(A)$ where A again represents a class label, X is the input feature vector for a pattern, $p(X|A)$ is the likelihood of the input data for class A and $P(A)$ is the prior probability for class A." In other words, for an unknown author, the class with the highest likelihood multiplied by the class prior probability is selected as the classification.

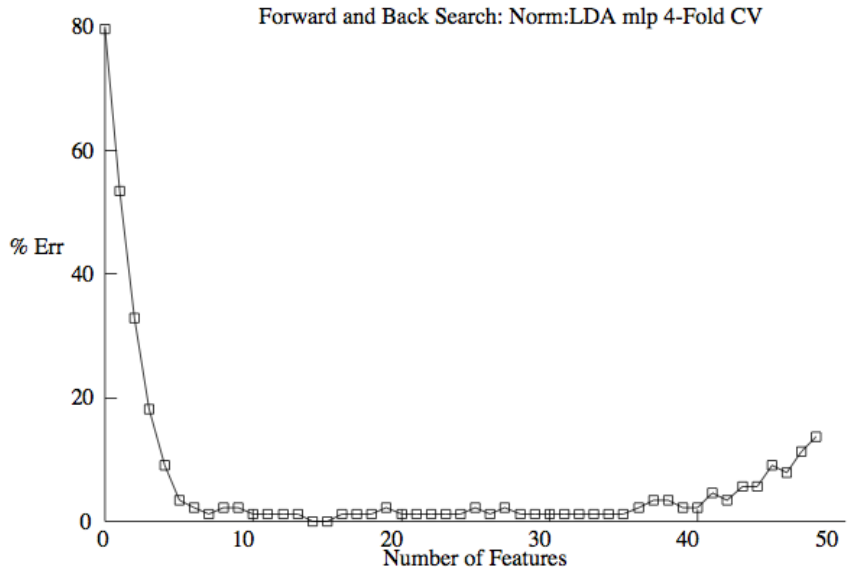

Figure 2.4 Classification Drops with Increase in Number of Features [24].

Krsul and Spafford [24] supply results showing an overall accuracy of 100 percent using 4-fold cross-validation as a search algorithm with forward search and LDA normalization using only a few metrics (Figure 2.5 [24]) However, as with the neural network classification, the error rates increased significantly when the data set was not normalized using LDA (Figure 2.6 [24]).

2.2.2.5 Learning Vector Quantizer

Learning Vector Quantizer (LVQ) is a Nearest Neighbor based algorithm. Nearest Neighbor algorithms use distance measures to classify unknown test cases by assuming that the smaller the distance between the features of a known case and unknown case, the more likely the test case matches the class of the nearest neighbor. Krsul and Spafford [24] obtained results of 78 percent accuracy using LVQ with LDA and 4-fold cross-validation.

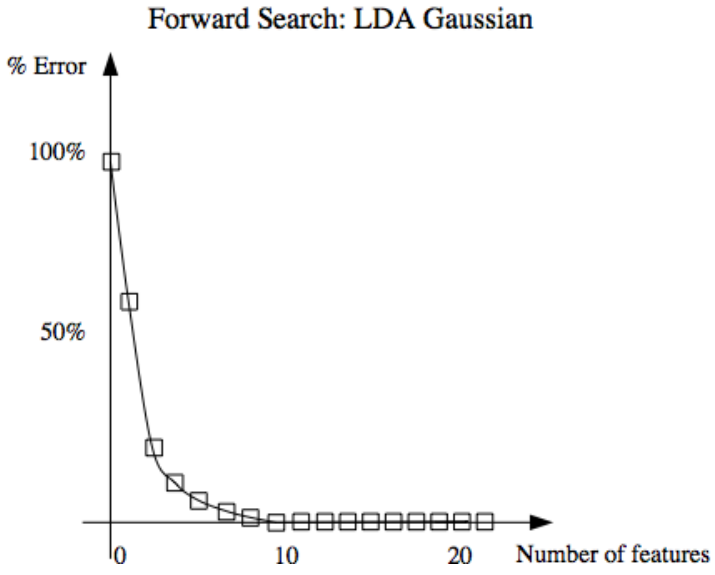

Figure 2.5 Gaussian Classification Accuracy Increases with FeaturesUsing LDA Normalization [24].

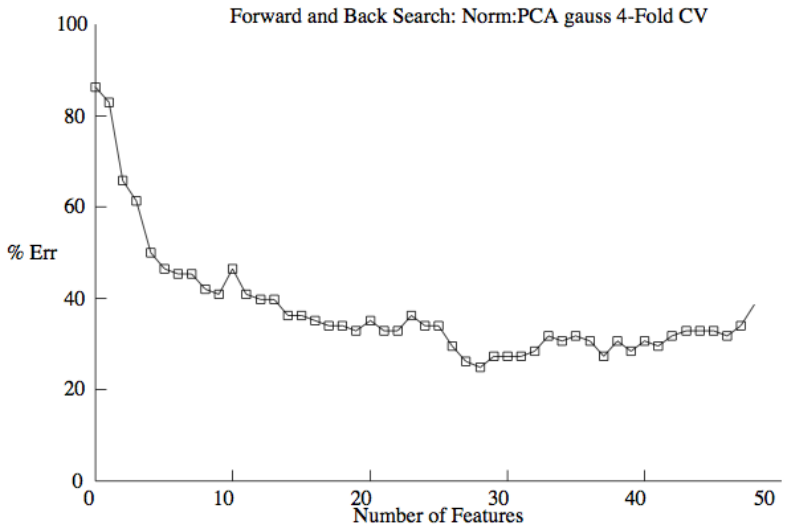

Figure 2.6 Gaussian Classification Using Principle Component Analysis Normalization (not as Accurate as LDA Normalization) [24].

2.2.2.6 Histogram

A histogram classifier is a likelihood, or probability classifier, that estimates the likelihood of each class by creating a set of histograms for the input feature. Each input feature dimension is divided into a number of bins, and the probability assigned to each bin is related to the number of training patterns that fall in the bin divided by the bin width. During testing, the probability for each input metric is multiplied to give an overall probability for each class. Krsul and Spafford's best classification using the histogram approach had an accuracy rate of 74 percent [24].

2.2.2.7 Binary Tree Classifier

Finally, Krsul and Spafford [24] used the BINTREE classifier provided by the LNKnet software. BINTREE is a rule-based classifier that partitions the input space into decision regions using threshold logic nodes or rules. The BINTREE classifier works well with problems with a small number of uncorrelated input features, but may have

difficulty on databases with high dimensionality [29]. Krsul and Spafford found BINTREE to be a poor classification technique with error rates of 30 percent [24].

In summary, Krsul and Spafford's technical report [24] was an in-depth attempt to expand upon Spafford and Weeber's work [7] to show that metrics derived from source code authorship programming styles can be used to classify programmers correctly using statistical analysis and machine learning techniques. At least two of the classification methods, MLP Neural Networks and Gaussian Classifiers, were able to classify unknown authors with 98 percent or better accuracy. Other methods were also able to classify authorship correctly with error rates about 70 percent.

However, some questions arise such as why these classification techniques were able to identify correctly the programmer who varied his style so dramatically over the short period of time. Krsul and Spafford [24] argue that two explanations are possible:

1. The programmer is indeed consistent in a way that is not evident to the human eye; therefore the consistencies can be detected by the classification methods but are not readily apparent upon visual inspection.
2. The programmer is inconsistent and the classification algorithm has assigned the author to a class of his own because of uniqueness.

In conclusion, Krsul and Spafford [24] showed through statistical analysis that programmers do indeed have stylistic tendencies when writing code. Programmers tend to reuse the same constructs, ones that help them complete their task more reliably and efficiently. Learned techniques and familiarity with constructs become part of an author's toolset. A level of comfort comes from knowing how to solve a problem in a particular fashion, and changes to approaches generally occur only when the situation dictates, or when a more eloquent solution is found. In addition, programmers even

organize sections of code in ways that are meaningful to their understanding. While some are neat and tidy with comments and even spacing, others are unsystematci, lack semantic meaning in variable or function creation, and leave out comments altogether.

2.2.3 Java and Fingerprints

This section describes another metrics-based approach. In 2003 Ding and Samadzadeh [30] used a metric selection process to extract author "fingerprints" from Java source code samples using discriminate analysis and the statistical software tool SAS. Out of the 56 extracted metrics, 48 were identified as contributing to authorship identification. However, the significance of the finding was that metrics associated with source code layout were more of a factor in determining authorship attribution than other metrics.

In the study [30], the data collected for the experiment was derived from three distinct sources. Programs were taken from students in the computer sciences classes at Oklahoma State University, a set of programs was collected from Internet shareware sites, and the third source for data was taken from a graduate associate who volunteered her programs to Ding and Samadzadeh.

They then took the Java source codes and parsed the files extracting the various metrics. The metrics used were adapted from Krsul and Spafford's earlier work [24], the metrics proposed by Gray and MacDonell [2, 22, 23]. In all, 56 metrics were extracted from 46 groups of programs. What Ding discovered is that not all of the proposed metrics contribute significantly to the authorship prediction. While some do, others may not.

Ding and Samadzadeh used two techniques to determine contributive variables from the metrics set. First they used one-way analyses of variance (ANOVA) for each individual variable, what he considered the "manual" method using the SPSS software package. Next, he used the statistical procedure called stepwise discriminant analysis (SDA) that is part of the SAS statistical package using forward stepwise analysis. After applying these procedures, he applied canonical discriminant analysis (CDA) with cross-validation using SAS to determine classification. CDA provides canonical variates, linear combinations of metrics that were derived from the metric variables. These variates summarize between class variations [30].

In general, similar metric sets were selected by both manual variable selection and SDA variable selection [30]. What was interesting in terms of selection was which metrics were included and excluded, and which variables were surprisingly highly correlated to a fingerprint. For example, the metric PRO5d [30], which measured the number of "switch" and cases" statements, was excluded because there were too many missing values, meaning few programmers used these structures. However, the metric FPC, files per compilation (or number of source files in a project), which implies separating a program into a number of source files, might be an important metric for determining a programmer's fingerprint.

Ding and Samadzadeh's results are presented in Table 2.2 [30]. The table shows the first 20 canonical variates that were used in the analysis. The authorship of 62.6-67.2 percent of Java source files was correctly assigned using the original metric values [30]. Using the canonical variates, the percentages were higher, up to 85.8 percent [30]. In addition, when source code authorship was classified at the project level, not the individual source code file level, the accuracies were higher. As shown in Table 2.2,

manual metric selection (ANOVA) versus automatic metric selection (SDA) had very similar results; but when canonical variates were applied, the classification results were affected.

The Java programs from Internet shareware sources and from the fellow graduate student were all correctly assigned although misclassification was common among the source codes from the computer science classes. They speculate that programming levels of the students or in class code examples may be to blame for the reduction in the between-class variations. This also implies that classification would be more effective if the diversity of the data sources increases. What is more interesting is that Dina and

Table 2.2 Classification Accuracy Using Canonical Variates [30]

Variables	Data sets			
	A	B	C	D
Selected metrics	62.7	67.2	62.6	66.6
CV1	19.8	17.7	18.6	16.1
CV1-2	41.0	39.0	41.3	38.2
CV1-3	52.1	51.0	54.3	54.1
CV1-4	57.5	60.6	57.3	61.9
CV1-5	64.1	67.7	65.1	66.2
CV1-6	66.4	72.1	69.0	72.6
CV1-7	70.7	76.1	71.5	75.8
CV1-8	71.6	76.8	70.7	78.1
CV1-9	71.7	78.7	72.9	81.6
CV1-10	70.6	79.5	73.7	82.0
CV1-11	70.8	78.2	72.3	84.4
CV1-12	72.6	80.2	74.5	85.2
CV1-13	76.0	81.1	79.2	84.4
CV1-14	76.5	80.2	79.7	85.8
CV1-15	76.9	82.0	79.6	85.8
CV1-16	77.1	81.2	80.0	85.4
CV1-17	77.6	81.6	77.8	84.9
CV1-18	77.0	81.6	78.2	84.4
CV1-19	77.0	82.5	78.4	84.9
CV1-20	76.9	81.9	78.8	83.9

Note: Classification accuracy was calculated as the percentage of correctly allocated samples.

Samadzadeh [30] found that metrics associated with layout played a more important role in the classification than style or structure metrics. Outstanding metrics were as follows [30]:

1. STY1c (percentage of open braces ({) that are the last characters in a line).
2. STY1g (average indentation in white spaces after open braces ({)).
3. STY1h (average indentation in tabs after open braces ({)).
4. STY3 (percentages of condition lines where the statements are on the same line as the condition).
5. STY4 (average white spaces to the left side of operators).
6. STY5 (average white spaces to the right side of operators).

2.2.4 Metric Histograms with Genetic Algorithms

In a case study, also focusing on metric-based approaches, Lange and Mancoridis [31] successfully identified authors by measuring the differences in histogram distributions for code metrics. The metric extraction approach is somewhat similar to that used by Ding and Samadzadeh [30]; however, where Ding and Samadzadeh use mostly scalar metrics derived from source code, the metrics of Lange and Mancoridis are formulated as histogram distributions.

An example histogram distribution would be a metric that measures the character length of each line of source code. The histogram representation would have an x-axis corresponding to every recoded line length while the y-axis point would represent the number of times a line of that length was exhibited in the code. The histogram would then be normalized by dividing the value of each y-axis point by the sum of all y-axis

points, which ensures that the sum of the y-values in the histogram is 1 [31]. Figure 2.7 [31] shows such a histogram.

The classification method proposed by Lange and Mancoridis [31] uses the nearest neighbor search using a general distance; in this case the general distance classifies the authors by measuring the smallest difference in histogram distributions from the unknown authorship source code with the known authorship code. Also, because the approach uses a nearest neighbor classifier, a ranked list of authors is produced in order of descending likelihood. Lange and Mancoridis uses the term precision [31] to define how far down the list the actual author of an unknown code sample is ranked. For example, if the algorithm correctly classifies the true author of a source code, then the author is at the top of the list and precision is 1. However, if the author is the second likeliest candidate, and is second on the list, then the precision is 2. A precision of 3 indicates that the true author was ranked behind two other authors, and so forth [31].

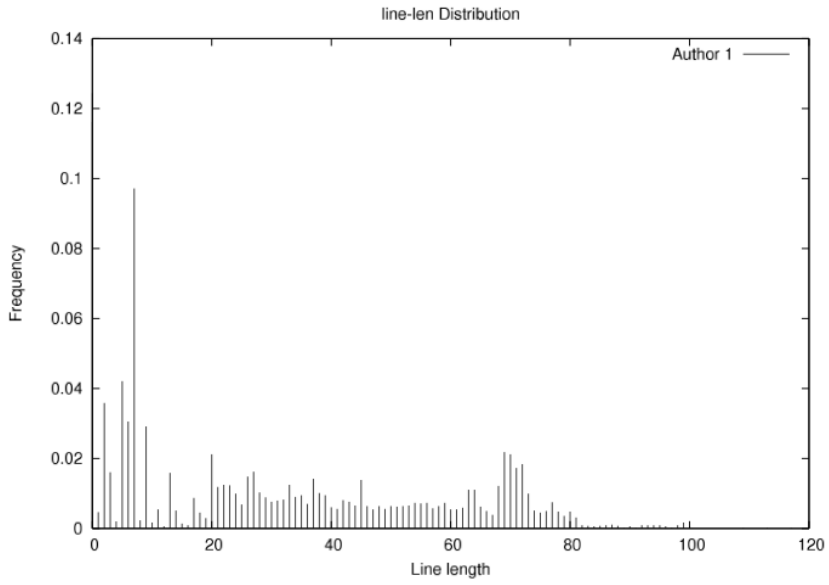

Figure 2.7 A Histogram for the Metric Line-Length [31].

The data for the study consisted of 60 open-source software projects written by 20 authors, each authoring 3 independent projects. The classification algorithm was tasked with classifying 40 projects (2 sets of 20 projects). Single metrics classified decently; for example, the line-length metric successfully classified 45 percent of the test data set. However, when all metrics were combined, the algorithm performed poorly, with a 70 percent error rate [31].

Combining sets of metrics to find the optimal combination of metrics is temporally intensive. Seventeen metrics result in $2^{18} - 1 = 262{,}143$ possible metric combinations [31]. Even with a typical runtime of about 30 seconds per execution, this process would require at least 65 days to complete on a single processor [31]; therefore constraints prevented the selection of no more than 18 metrics for the experiment.

To solve this problem, Lange and Mancoridis [31] decided to replace the exhaustive search by implementing a genetic algorithm solution. However, the approach was only moderately successful. They had problems getting the populations to converge after several days, even after putting into place measures to ensure that the population remained active despite converging to local maxima [31]. After modifying parameters, the algorithm converged to highly similar results after multiple iterations. Table 2.3 [31] shows the metrics and their results. The best 1-precision metric combination achieved 55 percent, while a 3-precision classified 75 percent of the projects.

2.2.5 Discretized metrics

Somewhat similar to the work of Lange and Mancoridis [31], in 2009 Shevertalov et al. [32] published a paper that focused on using genetic algorithms to discretize metrics to improve source code author classification. Discretization is the

process of partitioning a continuous space into discrete intervals [32]. In other words, it is a further classification, or reclassification, of already known metrics. For example, some developers may use verbose language to add comments to their source code. Instead of quantifying the size of their comments based on the number of characters, words, or lines of text, one can create three categories, short, medium, and long [32].

Table 2.3 Performance of Histogram Metrics Found by Gentic Alogrithm [31]

Precision	Success	Combination
1	11/20	inline-space inline-tab line-len line-words period trail-space trail-tab[2] underscore
2	14/20	brace-pos comment control-flow indent-space inline-space inline-tab line-len line-words period trail-tab underscore word-first-char word-len
3	15/20	brace-pos comment indent-space inline-space inline-tab line-len line-words period switch trail-tab underscore word-first-char

Of course the problem is in determining how to define the number of categories to be considered and the intervals within each category. For example, what constitutes short, medium, or long with regard to lines of text. Once the size and space are defined, is the discretization of the metric optimal? In previous work, Shevertalov, Stehle, and Mancoridis [33] demonstrated that discretizing large histograms into wider intervals when researching packet stream classification proved to be more effective.

The fundamental idea behind the approach is that most data mining problems have issues when it comes to data selection. Because there is a plethora of data to select; the question becomes which data is reliable as an indicator for classification. In other words, many metrics can be derived from source code, but which ones are useful? The discretization process used by Shevertalov, Stehle, and Mancoridis is an extension of the work of Dougherty Kohavi, and Sahami in machine learning [34]. Dougherty, Kohavi, and Sahami described the need for machine learning algorithms to have a discretized search space, and the effectiveness of various algorithms when provided categorical variables as opposed to continuous ones [34].

The approach of Shevertalov et al. taken in [32] is the same as other stylistically based metric approaches. Source code of known authorship was taken and run through a metric extraction process, histograms were constructed, and an author profile was derived. Next, four metrics were selected from previous works that produced the best results with identifying authorship [32]:

1. Leading spaces measure the amount of white space used at the beginning of each line. The x-axis value represents the number of the given white space characters at the beginning of each line.

2. Leading-tab measures the number of tab characters used at the beginning of each line. The x-axis value represents the number of the given tab characters at the beginning of each line.
3. line-len measures the length of each line of source code. The x-axis value represents the length of a given line.
4. line-words measures how densely the developer packs code constructs on a single line of text. The x-axis value represents the number of words on a line.

Taken from Kim and Han's work [35] on the use of genetic algorithms to perform discretization for a neural network to predict stock price indices, Shevertalov et al. applied the technique to help discretize the metric information for help with authorship classification [32].

In order for the GA to determine the fitness of a particular discretization, two things must take place. First, histograms need to be encoded so that break points are given in an effort to place beginning and ending values for the discrete interval, as shown in Figure 2.8, taken from [32]. (Shevertalov et al. used bits to represent breaks.) These breaks create the bins for the discrete histogram classification. Second, an evaluation function is required to assess the fitness of the newly discretized histograms to determine performance. Four parameters were used to calculate the fitness [32]:

1. Number of misclassifications.
2. Number of bins.
3. Distance from each classified entity to the correct class.
4. Distance from each classified entity to the incorrect class.

The evaluation function performs a nearest neighbor classification of the data after a representative histogram for each class is identified. The performance of the

nearest neighbor classifier is improved by either reducing the number of histograms in the learning set or by reducing the number of categories in each histogram [32]. Shevertalov et al. tested the technique using open-source code samples generated by 20 developers over 60 projects. The number of style metrics considered in the experiment was limited to four so that a comparison of results could be analyzed against undiscretized results from previous work.

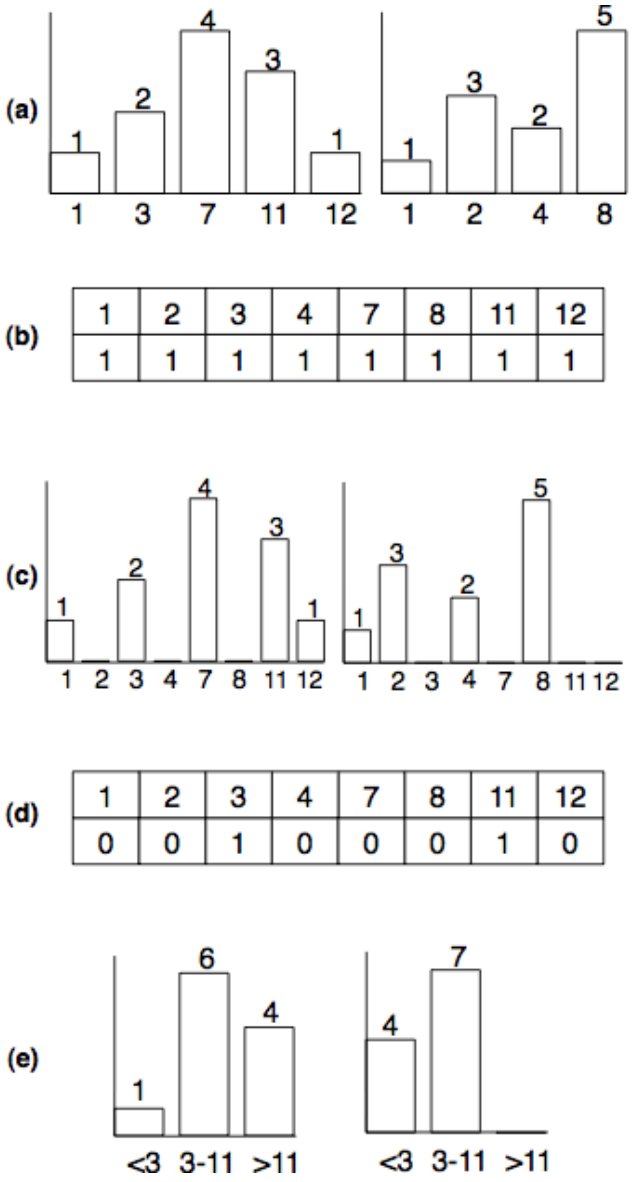

Figure 2.8 Process Demonstrating How the Encoding Can Be Used To Combine Multiple Histograms [32].

Note: (a) presents the original histogram. (b) illustrates the encoding where it is composed of all 1's and thus every bucket is its own bin. (c) demonstrates the results of the string encoding described in (b). (d) presents another sample encoding such that the result is three buckets: 0-2, 3-7, and 8-12. (e) demonstrates the results of the discretization described in (d) [32].

The testing was performed at two levels; individual source code files were classified as well as entire projects. When authorship was tested at the file level, the

discretized genetic algorithm approach outperformed nondiscretization with 53.3 percent correct classification to 46.1 percent [32]. At the project level, the discretized genetic algorithm approach had 25 percent error rate compared to 35 percent for nondiscretized [32].

2.3 Disadvantages of Metrics-Based Solutions

More recently, there have been new research attempts to move away from metric-based methodologies citing disadvantages [36]. Disadvantages cited are that software metrics are programming language dependent. Metrics used in Java may not be applicable in C or Pascal. Also, metric selection is not an elementary exercise. In [24], over 50 metrics are identified with a selection process that involves setting thresholds to eliminate metrics that contribute little to the classification model. The feature selection is not a trivial process, and usually involves setting thresholds to eliminate uninformative features [37]. These decisions can be extremely subtle, because although rare features contribute fewer signals than common features, they can still have an important cumulative effect [38]. This may have adverse effects on the analysis, thereby skewing results. In addition, multiple authors have commented on the difficulties of identifying the correct metrics and how dependent successful results are on their combination and identification. For example, Lange and Mancoridis [31] make a disclaimer in their abstract:

> Identifying a combination of metrics that is effective in distinguishing developer styles is key to the utility of the technique.
>
> Another conclusion we can draw is that if a combination of metrics identifies a developer's style strongly on one set of code, that same set will not necessarily identify the developer's style well on another set of code. To reduce the

> negative effect of project-specific rather than developer-specific style fingerprints, we now require no less than three independent projects per developer in our data set.

Krsul and Spafford [24] also comment on the metric selection process:

> This technique (discriminate analysis) works best with those metrics that show little variation between programs (for a specific programmer) and large variations among programmers. Unfortunately, analysis of the metrics collected shows that these two criteria are not necessarily correlated. Initially, we calculated the standard error by programmer for every metric, and eliminated those that showed large variations because they identify those style characteristics where the programmer is inconsistent. Surprisingly, most of the metrics that showed large variations among programmers were eliminated as well. The performance of our statistical analysis with the remaining metrics was discouraging, with only twenty percent of the programs being classified correctly.
>
> The step discrimination tool provided by the SAS program [SAS] should theoretically be capable of eliminating metrics that are not useful from the statistical base. Unfortunately, this tool was not helpful because it failed to eliminate any of the metrics from our set.
>
> The statistical analysis tools used provide little support for ranking the performance of individual metrics. The removal of any one metric from the analysis can have negative or positive effects, independent of the quality of the metric.
>
> Even though we are satisfied with our choice of metrics, it is possible that with them we would not be able to correctly classify a larger set of programmers successfully. Experience and logic tell us that a small and fixed set of metrics are not sufficient to detect authorship of every program and for every programmer. Also, by no means do we claim that the set of metrics we examined is the only one that might yield stable measurements.
>
> We do not expect that the metrics calculated for any given programmer would remain an accurate tag for a programmer for a long time, even though in our experiment….Further research must be performed to

> examine the effect that time and experience has on the metrics examined on this document.

In addition, Patrick Juola also mentions the overhead associated with metric-based approaches [8]. In the literary domain, a popular approach to authorship attribution is to identify a set of metrics/markers called function words. Using these function words, neural network approaches have also been applied to literary authorship with some success. However, they offer some of the same drawbacks of complexity, determination of function words (metrics), and sample size as mentioned by Juola in [8]:

> A similar question arises about the effort required to use any sort of tests. (Tweedie et al, 1994) provides a simple example; in their authorship tests, they painstakingly counted every occurrence of eleven function words in the undisputed corpora, normalized to appropriate density, and used a neural network to perform the actual determination…Even granting that the counting process can be automated, this procedure displays a fair degree of linguistic sophistication in the selection of which eleven word tokens to use (why is "his" a function word, and not "are" for instance?)
>
> By contrast, cross entropy of character distribution requires almost no pre-processing and can, if the (sample size) estimates are accurate enough, yield useful similarity results. The estimation techniques applied here is sufficient to yield accurate decisions about the language used in a document based on a database samples of only 100 characters…"

2.4 Nonmetric-Based Approaches to Software Forensics

As a result, the focus in other research efforts, such as [2] and [30], was into the metrics selection process rather than into improving the effectiveness and efficiency of the proposed model [36]. However, more recently research has been focused on identification techniques that are not metrics oriented; techniques that are more efficient

and less costly to implement. In this section, various nonmetric-based approaches are discussed along with the cross-entropy approach applied to literary works at the end.

2.4.1 Abstract Syntax Tree

In [39], Pellin uses the statistical machine learning technique, abstract syntax trees (AST) with support vector machines (SVM), to test authorship identification of source code. (An SVM is a set of related supervised learning methods used for classification and regression. Given a set of training examples, each marked as belonging to one of two categories, it will predict classification for an unknown or new example [40].) The classification problem is framed as a binary classification between two authors where two similar programs, written by different authors, are used to train the classifier algorithm. This classifier is then given an unknown, unlabeled function, and tries to determine which programmer is the true author.

Pellin uses a SVM [39] classification technique originally implemented by Moschitti [41] where traditional SVM methods that operate on flat feature sets in the form of vector numbers are instead transformed so that the kernel can operate on data in a tree structure.

An abstract syntax tree is a finite tree, in which the internal nodes are labeled by operators, and the leaves of the tree are operands. Figure 2.9 [39] shows an example tree applied to a simple Java code line. The Java programming language was used in the experiments, arguing that many Java source code examples are freely available on the Internet. Pellin also argues that AST is a natural way to view the program because AST is unaware of the spacing in the original text [39]. While the author of this work argues that spacing is a metric that has shown to contribute to authorship identification using a

metric-based approach, Pellin says the use of pretty printers would destroy any trace of authorship present in the source code with respect to spacing, which is also accurate. However, in most circumstances this author is skeptical that a malicious code author would perform this operation on a regular or even irregular basis.

When applied, the AST method breaks down the structure of the program at the method (function) level. Therefore, each AST in the training set represents a function. Because each function becomes a tree, there are two derived benefits: (1) decomposition at the function level reduces the complexity and size of the source code into measurable chunks, and (2) dividing the source code by the number of methods increases the number of documents in the collection. At the heart of the approach, the algorithm assumes that programmers have certain styles and that style will be reflected by the AST in the form of repeated substructures that represent repeated patterns by the programmer. Programs that have a higher number of matching subtrees (meaning the two trees are more similar) are more likely to have been written by the same author [39]. Classification takes place when an SVM kernel machine designed to operate on structured tree data parses the ASTs. Kernel functions provide a method for expressing the similarity of two objects without explicitly defining their feature space [41]. The algorithm is polynomial in time even though the number of sub-trees is exponential in the size of the subtree. The full details are defined in [39] and [42].

In the experiment, Pellin [32] selected pairs of open-source Java programs from the Internet. The functionality of each pair is similar, such as two open-source implementations of a Java web mail client, which helps minimize the differences in the programs not associated with authorship differences. The source files are then read into the Yaxx [43] parser where a tree is output in XML format.

```
System.out.println("Num=" + (5 * 6));
```

Abstract Syntax Tree

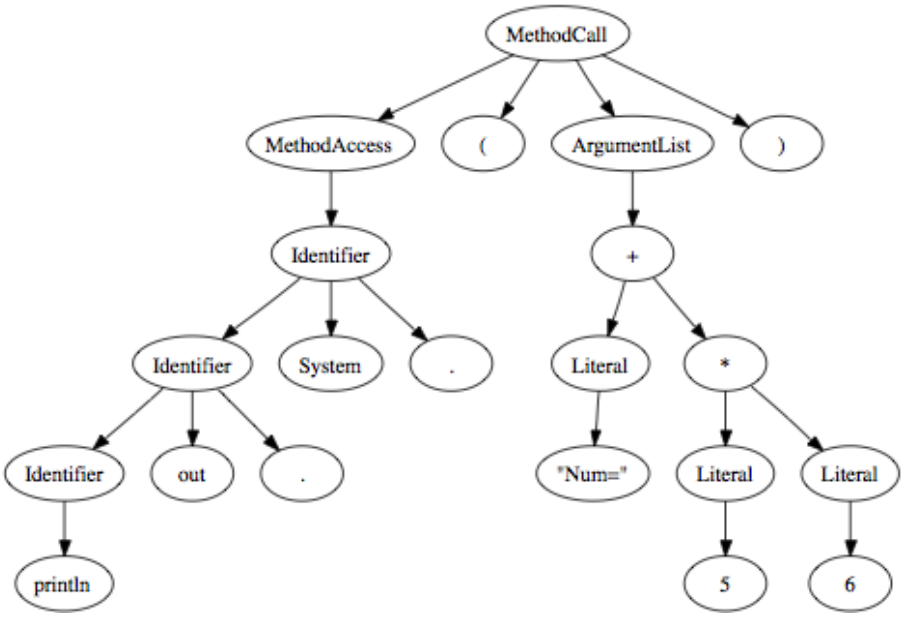

Figure 2.9 An AST for a Fragment of Source Code along with Its Abstract Syntax Tree.

Note: Some Single Parent, Single Child Internal Nodes Have Been Trimmed for Readability [39].

When applied, the AST method breaks down the structure of the program at the method (function) level. Therefore, each AST in the training set represents a function. Because each function becomes a tree, there are two derived benefits: (1) decomposition at the function level reduces the complexity and size of the source code into measurable chunks, and (2) dividing the source code by the number of methods increases the number of documents in the collection. At the heart of the approach, the algorithm assumes that programmers have certain styles and that style will be reflected by the AST in the form of repeated substructures that represent repeated patterns by the programmer. Programs that have a higher number of matching subtrees (meaning the two trees are more similar) are more likely to have been written by the same author [39]. Classification takes place when

an SVM kernel machine designed to operate on structured tree data parses the ASTs. Kernel functions provide a method for expressing the similarity of two objects without explicitly defining their feature space [41]. The algorithm is polynomial in time even though the number of sub-trees is exponential in the size of the subtree. The full details are defined in [39] and [42].

In the experiment, Pellin [32] selected pairs of open-source Java programs from the Internet. The functionality of each pair is similar, such as two open-source implementations of a Java web mail client, which helps minimize the differences in the programs not associated with authorship differences. The source files are then read into the Yaxx [43] parser where a tree is output in XML format.

Next the XML tree is dissected at the method level while the identifying names of the methods are removed from the tree. Pellin [39] argues that the names are removed in an attempt to make the trees more general, allowing the classifier to focus on the structure of the source code and method implementation. Because the algorithm is comparing only two source code projects, the data set is fairly small; therefore 10-fold cross validation was used to measure the accuracy of the classifiers.

The classification accuracy for the approach ranges from 67 percent to 88 percent [39]. However, it appears that the classification accuracy is highly sensitive to the data set. The closer in style that two authors are, the harder it is to find a good separation boundary between the two classes [39]. In addition, as expected, accuracy increased as the number of training examples increased ,but ten times more training data were required to get just a 5 percent increase in classification accuracy [39].

2.4.2 N-grams approaches

N-grams have successfully been used in natural language processing to help with speech recognition, language modeling, spelling correction, character recognition, and text authorship attribution. The underlying mathematics of the N-gram was first proposed by Markov (1913), who used what are now called Markov chains (bigrams and trigrams) to predict whether an upcoming letter in Pushkin's *Eugene Onegin* would be a vowel or a consonant [46]. Markov classified 20,000 letters as V or C and computed the bigram and trigram probability that a given letter would be a vowel given the previous one or two letters. In 1948, Shannon applied n-grams to compute approximations to English word sequences. Based on Shannon's work, Markov models were commonly used in modeling word sequences by the 1950s [46]. However, in the late 1950's, Noam Chomsky argued that finite-state Markov processes were not sufficient for modeling natural language and should be used only as a heuristic. These arguments led many linguists and computational linguists away from statistical models altogether [46] until the late 1970's and early 1980's.

An n-gram is an n-contiguous sequence and can be defined on the byte, character, or word level [36, 47]. The n-gram process starts by selecting a size for n. Next, starting at the beginning of the document, the first n-gram sequence is assigned by taking the first n characters in length. The process is repeated by moving one character toward the end of the document, sliding the n-gram window along the way until the end of the document is reached. During each move, more n-grams are identified and tabulated. The output of the process results in a table that displays the n-grams found in the document along with their frequency of occurrence. This table, which may or may not be normalized, is treated as the author's profile. For clarification, consider the following small example:

Suppose there is a string, 'ABCDEFGHIJKLMNOPABC' and we wish to find the n-grams for size n = 1, n = 2, and n = 3. For n = 1, an output table will be produced that shows the frequency of 'A' = 2, 'B' = 2, and 'C' = 2, since the letters 'ABC' appear both at the beginning and end of the string, with all other letters having a frequency of 1. When n = 2, the n-gram sequence 'AB' will appear twice, as well as the sequence 'BC', while all other pairs of letters will have a frequency of 1. Figure 2.10 shows output tables from this string using Keselj's perl package [47] Text::Ngrams.

One such approach is the Source Code Author Profiles (SCAP) method presented by Frantzeskou et al. [44]. The SCAP approach uses low-level nonmetric information to perform classification. Because of its low-level nature, it is also programming language independent. SCAP utilizes the byte-level n-gram approach popularized by Keselj et al. [45] where byte-level n-grams are utilized to establish and assess code against author profiles.

Using the n-gram SCAP approach [48] for software forensics, all source code files known to belong to a particular author are concatenated into one source code file. Next, the set of the L most frequent n-grams are extracted into a table and assigned as the author's simple profile. Then, test author profiles are generated from source code of unknown authorship for each test case. The test case profiles are then compared to the author profiles and a similarity distance is calculated. In SCAP, the similarity distance is given by the intersection of the two profiles:

$$SP_A \cap SP_T| \tag{2.1}$$

or, in other words, selecting the profile that has the largest number of n-grams in common.

```
2-GRAMS (total count: 18)
FIRST N-GRAM: A B
 LAST N-GRAM: B C
------------------------
A B     2
B C     2
C D     1
D E     1
E F     1
F G     1
G H     1
H I     1
I J     1
J K     1
K L     1
L M     1
M N     1
N O     1
O P     1
P A     1
```

```
3-GRAMS (total count: 17)
FIRST N-GRAM: A B C
 LAST N-GRAM: A B C
------------------------
A B C   2
B C D   1
C D E   1
D E F   1
E F G   1
F G H   1
G H I   1
H I J   1
I J K   1
J K L   1
K L M   1
L M N   1
M N O   1
N O P   1
O P A   1
P A B   1

END OUTPUT BY Text::Ngrams
```

Figure 2.10 Output of n-grams for 'ABCDEFGHIJKLMNOPABC'

The n-gram approach has some advantages over other metric-based approaches. First, it is independent of programming language [48], and it is not affected by the absence of comments by the programmer. Also, the algorithm does not require numerous training examples to calibrate the model for accuracy since it generates one profile per author. Obviously, the more source code available to create the profile, the more accurate the profile; however, decent results have been achieved where only a few source code samples were obtained [48]. This is important for software forensics investigators who may not have obtained enough evidence in the data collection process.

2.4.3 Cross-entropy

Cross-entropy is a nonmetric-based approach, which to the author's knowledge has not been applied to source code authorship studies. In information theory [10] entropy is simply a measure of the unpredictability of a given event, given all relevant

background information that could be brought to bear. Cross-entropy is a measure of the unpredictability of a given event, given a specific (but not necessarily best) model of events and expectations [10]. Cross-entropy is a technique proposed and applied by Juola [8-10] to literary authorship analysis studies with the main focus of the algorithm centering on "match length within a database." It can be treated mathematically as a "distance" between two documents, where a low distance describes two similar documents. It was developed by Wyner [14] and is outlined in the following section.

2.4.4 Cross-entropy Theory

Natural language can be very predictable. For example, most English readers will guess correctly the next letters in the following phrase "Probability and Statis~~~~." This is true because readers understand the rules of phonetics, language, and context. This notion of predictability, as well as the associated concepts of complexity, compressiveness, and randomness, can be mathematically modeled using information entropy [9]. C. E. Shannon introduced the idea of information entropy in 1948 [11]. Entropy can be described as the amount of information, usually measured as yes/no bits, that is required to describe messages from a message source. The entropy of a message source increases as the messages become less predictable, typically because the set of possible messages becomes larger, or the distribution of the messages becomes more varied. Below is Shannon's equation [11]:

$$H(P) = -\sum^{N} p_i \log_2 p_i \tag{2.2}$$

where P is the probability distribution of a source capable of sending any of the messages 1, 2,..., N, each with some probability P [9].

An important question here is against what is the predictability of the distribution measured? For the source code samples the actual probability distribution is not known; therefore cross-entropy can be used. This is an important point, especially if the size of the corpora is limited, which is often the case in software forensic investigations. The distribution of the messages (where messages are the samples taken from the source code corpora) needs to be estimated, as accurately as possible,. Cross-entropy is useful when the actual probability distribution *p* that generated some data is not known. It allows the use of some *q*, which is a model of *p* (i.e., an approximation to *p*) [46]. The second term in Equation 2 is a measure of the efficiency of the representation of message *i* (obviously, more frequent messages should be made shorter for maximal efficiency, an observation often attributed to Zipf [18]), based on the author's estimate of the frequency with which *i* is transmitted. Therefore, Equation 2 can be generalized to

$$\hat{H}(P,Q) = -\sum^{N} p_i \ -\log_2 q_i \tag{2.3}$$

where *Q* is a different distribution representing the best estimate of the true distribution *P*. This value (called the cross-entropy) achieves a minimum when $P = Q$, and $H(P, P) = H(P)$ [9]. From this, a measurement of similarity between two different sources can be estimated by focusing on the distributional parameters and calculating their cross-entropy.

Based on this, Wyner [14] has described a simple entropy estimation technique that uses the concept of "match length within a database." Wyner defines the match length $Ln(x)$ of a sequence ($x1, x2,..., xn, xn + x,...$) as the length of the longest prefix of the sequence ($xn + x,..$) that matches a contiguous substring of ($z1,z2,... ,xn$), and proves that this converges in the limit to the value $(\log n)/H$ as n increases [9].

The extent to which any literary similarity exists between two documents, whether similarity in author, topic, genre, or even characteristics of the author, will be reflected in a lower distance [10].

Cross-entropy [8, 10] also has two important operations:

1. Choosing the optimal size for the sliding window, which will become the profile.
2. Calculating the longest average prefix between the profiles and the test object.

To measure this distance, a sliding window of size n is applied to the text/characters of the document. The information inside the sliding window becomes the database (or profile) against which a test source code document of unknown authorship is compared. The goal is to find the longest continuous prefix that matches what is in the current database when comparing the documents. The longest prefixes are then averaged over the sliding window. The longer the length of the average prefix match, the more similar the documents are. The document that has the highest average prefix after making all comparisons is chosen as the solution. The following is a simple example [9, 10] .

Consider the phrase:

HAMLET: TO BE OR NOT TO BE THAT IS THE QUESTION

while setting n, or the size of the window, at 21. The database becomes the sequence "HAMLET: TO BE OR NOT" (length 21), and the phrase "TO BE THAT IS THE QUESTION" is the remaining data. The prefix "TO BE" exactly matches the contiguous substring starting at the eight characters with a length of seven characters within the database, but the prefix "TO BE T" does not match a continuous substring contained in the database. Therefore the longest match length at this point for L_{21} is seven.

By using this approach, an estimate of entropy can be measured. One can slide a block of n observations along the sequence and calculate the mean match length L, or L averaged over each step, which will give the entropy estimate. Continuing, after the window slides forward to modify the database, L_{21} becomes the string "AMLET: TO BE OR NOT TO BE THAT IS THE QUESTION," then "MLET: TO BE OR NOT TO BE THAT IS THE QUESTION W," and so on.

The testing material for the work of Juola and Baayen [10] consisted of Dutch writings from students at the University of Nijmegen. The students, all of whom were undergraduates in Dutch literature, were asked to write three papers from different genres. In the experiments, the cross-entropy approach faired well with a successful identification rate of about 87 percent.

CHAPTER III

APPLICATION OF CROSS-ENTROPIC APPROACHES TO SOURCE CODE CORPORA: EXPERIMENTAL DESIGN, FOCUS, AND STRUCTURE

3.1 Experimental Design and Framework

The experimental design of this work will follow the paradigm of research studies mentioned in Chapter II. In most source code authorship attribution studies the approach is the same. A corpus is constructed where source code is acquired from open-source sites on the Internet, from computer science students at universities, or from source code repositories where the researcher has access. After the corpus is established, the experiment is conducted, the researcher applies his or her technique to data sets, and the results are tabulated.

Of course, one drawback with this approach, as with authorship attribution studies aimed at literary works, is the lack of a common corpus for testing, which would provide a solid basis for all approaches to truly determine the best method or practice. The sheer number of methods proposed and the sheer number of test corpora developed make it difficult to identify any clear "best practices" or particularly accurate techniques [49]. In the literary arena, Juola has identified this as a shortcoming and has proposed a “reusable” test corpus, the Java Graphical Authorship Attribution Program or JGAAP [49], as a system for comparative evaluation of authorship attribution techniques for literary works. Unfortunately, there is no such effort for source code authorship attribution studies (although there are many digital forensic corpora, such as disk images,

anonymous emails, network log files, etc.). Therefore the author has constructed several corpora from available source code repositories consisting of professional computer programmers and students from computer science classes.

The remainder of Section 3.1 describes the experimental design and framework by detailing the various corpora, their construction and attributes, anonymization techniques, file outputs, and mechanisms for filtering and analysis of results. The remainder of Chapter 3 focuses on the research questions to be addressed by this body of work with experiment execution, results, and discussion following in Chapter 4.

3.1.1 Anonymous Source Code Corpora

In order to compile a corpora data set for testing, this work proposes using student source code files from Mississippi State University Computer Science classes and professional programmer source code files from the U.S. Army Engineer Research and Development Center. The author of this work understands the importance of privacy; therefore a mechanism has been devised such that the original author will be identified anonymously using a mapping to a non-descript identifier, such as a unique ID number. (For example, student "John Doe" was renamed student "123" to protect privacy information to mitigate unintended consequences that could arise from subject involvement. For more information see Section 3.1.2.1.)

The author of this work also realizes that using student data does infringe upon privacy and ethical issues; therefore, the training and authorization offered by Mississippi State University's Office of Regulatory Compliance-Institutional Board for the Protection of Human Subjects in Research were sought. Upon completion, the author will work in

coordination with Mississippi State faculty to develop an anonymous source code test corpora.

3.1.2 Corpora Construction, Structure, and Attributes

Five distinct corpora were constructed for this research effort and applied during the experimental execution phase detailed in Chapter 4. Four corpora were constructed from different programming courses at Mississippi State University. One corpora was constructed from professional programmers at the Engineer Research and Development Center's Information Technology Laboratory. The remainder of this section will detail the attributes associated with each corpora.

3.1.2.1 Student Corpora Construction Overview and Anonymization

Three instructors who were offering courses at Mississippi State University's Computer Science and Engineering Department were contacted and agreed to participate in this research effort.

The Course 1 corpora comprised primarily electrical engineering students with the purpose of providing an introduction to software programming using the C++ programming language. The Course 2 corpora comprised primarily computer science students using the Java programming language. The Course 3 corpora comprised primarily computer science students in an introductory programming course using the Python programming language and C++. The Course 4 corpora is the same course as Course 2 with different students. It also comprised primarily computer science students using the Java programming language.

In order to safeguard participant privacy, all course instructors were provided by the author of this work with an anonymization application written in either C#, or Java

for non-Windows users, which can be found in Appendix B. The anonymization application has the following instructions and works in the following manner with the listed constraints:

1. The user will place each student assignment in an individual folder. (For example, create a directory for assignment number1 and put all source code files for assignment number1 in that directory. Only source code files are needed. Please disregard header files, solution files, binaries, etc.)
2. Next, create a text file containing the students' names, each name on a separate line. Make sure that each student's name exactly matches the name in the source code file.
3. The student's name must be located on the very first line in the source code file. No blank lines, comments, //, */, etc., just the name as it appears in the student's name text file.
4. When processing, make sure to update the assignment field in the application before processing the next assignment.

In practice, the anonymization application takes the text file from step 2, creates an anonymous identifier for each student in an array type structure, opens the first line of each student source file given a directory, reads out the author name and performs a lookup for the current student from the array, replaces the first line containing the student name with the nondescript identifier within the source code, and processes the next file in the directory until all files are complete. This processs anonymizes the source code for the purposes of this experiment by replacing all identifying information with a nondescript unique identifier.

3.1.2.2 Student Corpora - Course 1 Structure and Description

As mentioned previously, Course 1 comprised primarily electrical engineering students. The purpose of the course is to provide an introduction to software programming using the C++ programming language. The Course 1 corpora comprise 81 source files from 17 different student authors. Most student authors submitted five programming assignments, with the exception of a few, leading to a total of 81. Using pairwise comparison, this led to a total of 6480 cross-entropy experiment runs. (Defined here, a pairwise comparison means each source file will be compared against all other source files, i.e., matched "head-to-head," making the number of comparisons equal to $n * (n-1)$ where n is equal to the number of files in the experiment, or in this case, all the files in the corpora.) In addition, each run is computed for 20 different window sizes per run for a total of 129,600 comparisons.

3.1.2.3 Student Corpora - Course 2 Structure and Description

Course 2 comprised primarily computer science students. The purpose of the course is to provide an introduction to software programming using the Java programming language. Most student authors submitted five programming assignments, with the exception of a few, leading to the total of 87 source files. Using pairwise comparison, this led to a total of 7482 cross-entropy experiment runs. In addition, each run was computed for 20 different window sizes per run for a total of 149,640 comparisons.

3.1.2.4 Student Corpora - Course 3 Structure and Description

As mentioned previously, Course 3 comprised primarily computer science students and serves as an introductory programming course using the Python

programming language and C++. The Course 3 corpora comprised source files from 20 student authors who submitted files for five different programming assignments, three of which were written in Python. As in the Course 1 and 2 corpora, some students' assignment files were not submitted, giving a total of 49 source code files. Two of the assignments were written using C++. In addition, the C++ assignments contained an extra credit assignment for which some students submitted a file while others did not, for a total of 51 files in the C++ corpora. (Note: In the experimental execution for these source code samples, which will be presented in Chapter 4, cross-entropy will be applied only to source code samples from the same programming language. No cross-language experiments will be conducted.)

For the Python experiments, using pairwise comparison for 49 files led to a total of 2352 cross-entropy experiment runs. In addition, each run was computed for 20 different window sizes per run for a total of 47,040 comparisons. For the C++ experiments, using pairwise comparison for 51 files gave a total of 2550 cross-entropy experiment runs. In addition, each run was computed for 20 different window sizes per run for a total of 51 comparisons.

3.1.2.5 Student Corpora - Course 4 Structure and Description

The Course 4 Corpora comprised primarily computer science students, 15 who freely submitted code. The purpose of the course was to provide an introduction to software programming using the Java programming language. Course 4 was the same class as Course 2, just a different section; therefore the programming assignments were the same, just from a different set of students. (Note: It will be interesting to see how these experiments fare when compared with Course 2 experiment results.)

There are a total of 72 source files in Course 4. Once again using pairwise comparison, this led to a total of 5112 cross-entropy experiment runs, computed for 20 different window sizes per run, giving a total of 102,240 comparisons. Combined with the 390,000 comparisons from the previous student corpora experiments, this sums to almost 500,000 comparisons.

3.1.3 Professional Corpora Structure and Description

In addition to student corpora, a professional corpus was used for experimental testing. To provide data points for the professional corpora, the U.S. Army Engineer Research and Development Center's Information Technology Laboratory has a number of source code repositories contributed to by many developers for a variety of software systems, whose file sizes and lines of code range from small to extremely large.

Source code files from five authors, all of which have at least 2 years programming experience, were selected with their consent from the source code repository to comprise the professional corpus. Each author contributed 5 files for a total of 25 files; the author was the sole author of the code file in question. This is important because in a large organization that focuses on software development, source code files can have co-authors at different points in the development cycle. To ensure single authorship, each participant was asked to provide the five source code files from any development project where he or she could verify sole authorship for the file submitted. In addition, because the files were submitted at the discretion of the programmer, the functional purpose of the files was likely dissimilar, unlike the student corpora where homework assignments defined the same functional objectives, which allows for different experimental comparisons.

The professional programming corpora was composed of 25 files; therefore, in order to compare every file against all others, a total of 600 comparisons were made (excluding comparison against itself) or 25 * (25-1) comparisons. In addition, because cross-entropy can use variable window sizes, each pairwise cross-entropy comparison was executed 20 times. The experiment compared the files using variable window sizes from 5 to 100 in increments of 5 for a total of 20 runs per file, which resulted in 12,000 runs.

With so many cross-entropy comparisons, all runs from each experiment had to be sorted and analyzed in order to find the best matching author, meaning the comparison containing the highest mean match length ratio determined by the cross-entropy algorithm. The next section describes this process.

3.1.4 Sorting, Filtering, and Analysis

Because of the high number of comparisons at varying window sizes, a postprocessing mechanism was needed to sort and filter the results to find the most suitable candidate author (in this case the author with the highest score after cross-entropy computation) for an experiment comparison. This section describes that process.

The output of a single pairwise comparison, at window size n, is the mean match length ratio. The mean match length ratio is equal to the sum of the longest prefix found over a processing event, where a processing event is defined as a file-to-file comparison using the cross-entropy algorithm approach for a particular window size, divided by the index. The index is defined as the number of window slides needed to process a file from beginning to end, which is then divided by the current window size for the comparison in question, as shown below.

(Sum of All Longest Found/Index) / Window Size

For sorting, the output file for a cross-entropy authorship experiment will look similar to outputs shown in Table 3.1. Each column represents a file-to-file comparison, or experiment run, where a known author's code (Author1) is compared to an unknown author code (Author 2 and Author 3). In practice, each column would actually be stored in a separate file.

Each row or line within the output file is a comma-delimited set where the first value is the window size for the comparison and the second value is the mean match length ratio, which is calculated by cross-entropy for the comparison. The column headings in Table 3.1 show the known authorship file output compared against unknown authorship source files by different authors for an experiment run.

To determine the best candidate author for a given window size, a sorting routine was constructed. This routine opens each file comparison output file, parses the results for a given window size (a particular line in the file), and stores the value in a sortable array/hash table, where the name of the experiment run comparison is the key. The sorting routine will continue this process at every window size, until it is finished processing at window size 100. Once finished, the sorting routine will take the top candidate author, per window size, and create a single results/analyis output file indicating the candidate author, along with the source code file name by the author, with the highest match length determined by the cross-entropy algorithm. Table 3.2 shows an example sorted output file after processing. (A line can be read as "run unknown test file for class 1233, assignment 4, student 3 - compared to – candidate file class 1233, assignment 4, student 12, window size 5, mean match length 0.839".) Note how window size (the second value after the comma per line) affects classification. This particular

example shows that for window sizes 20 through 50, student number three, assignment 4, most closely matched student number 3, assignment 5, i.e., a correct authorship classification at those window sizes.

Table 3.1 Example Output from an Experiment Run.

Author 1 Vs. Author 2	Author 1 Vs. Author 3	Author 1 Vs. Author 4	…
5,0.6002493765586034	5,0.6239888423988843	5,0.6316410861865407	
10,0.378168130489335	10,0.38523442967109867	10,0.3986935866983373	
15,0.2553872053872054	15,0.26465355805243446	15,0.2765432098765432	
20,0.19193138500635326	20,0.19929527836504582	20,0.2076923076923077	
25,0.15416879795396418	25,0.1596039603960396	25,0.16681983071342202	
30,0.12831402831402833	30,0.13288384196829903	30,0.139213300892133	
35,0.10921539600296076	35,0.11412291412291412	35,0.11893687707641197	
40,0.09563233376792699	40,0.09992852037169406	40,0.10301724137931034	
45,0.08533100029163021	45,0.08895265423242468	45,0.08993528844829961	
50,0.0770673712021136	50,0.08012958963282937	50,0.0812718204488778	
55,0.0702852998065764	55,0.07280609563846557	55,0.07414166761720087	
60,0.06443551985720661	60,0.06678752719361856	60,0.06818181818181818	
65,0.05948579722164628	65,0.06168402194603068	65,0.0630827876062946	
70,0.05497189377786393	70,0.05736199519983303	70,0.058604311289733285	
75,0.051111111111111114	75,0.0535483870967742	75,0.05462033462033462	
80,0.047300550206327376	80,0.05023914643119941	80,0.051068652849740936	
85,0.04433762424637445	85,0.04731080024328786	85,0.04776439911036122	
90,0.04201146753448008	90,0.04473272382835022	90,0.044634004082822976	
95,0.039976345357776465	95,0.04243421052631579	95,0.04232774803587568	
100,0.03792079207920792	100,0.04035847647498133	100,0.04035904255319149	

Note That Each Column Is an Actual Output File

Once sorting and analysis were completed, the results were tabulated and examined for patterns or trends with respect to such variables as file size, programmer experience, and functional/objective goals for a particular code.

Table 3.2 Example Sorted Results File.

Author 3 – Source File 4 Results
runcse1233-4-3.c-VS-cse1233-4-12.c.txt,5,0.8393939393939395
runcse1233-4-3.c-VS-cse1233-4-9.c.txt,10,0.65015625
runcse1233-4-3.c-VS-cse1233-4-15.c.txt,15,0.5478945297127116
runcse1233-4-3.c-VS-cse1233-5-3.c.txt,20,0.4749406175771972
runcse1233-4-3.c-VS-cse1233-5-3.c.txt,25,0.4185902031063321
runcse1233-4-3.c-VS-cse1233-5-3.c.txt,30,0.3712339743589743
runcse1233-4-3.c-VS-cse1233-5-3.c.txt,35,0.32962515114873037
runcse1233-4-3.c-VS-cse1233-5-3.c.txt,40,0.292396593673966
runcse1233-4-3.c-VS-cse1233-5-3.c.txt,45,0.2597579219366245
runcse1233-4-3.c-VS-cse1233-5-3.c.txt,50,0.2311576354679803
runcse1233-4-3.c-VS-cse1233-4-9.c.txt,55,0.208582995951417
runcse1233-4-3.c-VS-cse1233-4-15.c.txt,60,0.1907522859517872
runcse1233-4-3.c-VS-cse1233-4-15.c.txt,65,0.1767976064086478
runcse1233-4-3.c-VS-cse1233-4-15.c.txt,70,0.1643939393939394
runcse1233-4-3.c-VS-cse1233-4-15.c.txt,75,0.15322321050402374
runcse1233-4-3.c-VS-cse1233-4-15.c.txt,80,0.1444693094629156
runcse1233-4-3.c-VS-cse1233-4-15.c.txt,85,0.13661897191308958
runcse1233-4-3.c-VS-cse1233-4-15.c.txt,90,0.12928900402993668
runcse1233-4-3.c-VS-cse1233-4-15.c.txt,95,0.12239072256913469
runcse1233-4-3.c-VS-cse1233-4-15.c.txt,100,0.11585301837270341

Note How Window Size Affects Accuracy and Therefore Candidate Author Selection.

3.2 Research Questions and Research Focus

The focus of this research is to examine the effectiveness of the cross-entropy approach when applied to source code authorship attribution. More formally, the basic research question being examined in this work is as follows:

Can a cross-entropy approach be used to predict source code authorship? Cross-entropy has been shown to identify authors in literary works. Will cross-entropy approaches taken in literary document classification and authorship identification fare

similarly when compared with cross-entropy approaches applied to source code authorship identification?

The purpose of this question is to determine the validity of cross-entropy as a technique to accurately predict source code authorship. The approach is already known to be successful in literary works, with a classification rate of about 73 percent [10]; now the approach will be explored as a solution to authorship attribution in programming language source code corpora. To explore the validity of the technique, the experimental design framework and corpora described in Section 3.1 will be applied to the cross-entropy approach to gauge the accuracy of the algorithm.

Also related to this research question is the assumption that experience should help the algorithm discern authors because distinctive traits are developed with experience, as older habits are tried, true, and reliable. Therefore, the first research hypothesis to be tested is as follows:

The cross-entropy approach will more accurately identify experienced programmers when compared with less experienced programmers.

Obviously, the professional corpora and student corpora, as well as assignments later in the semester, will be used to answer this research question. It is assumed that students later in the semester will be more experienced and will have developed an identifiable style. In fact, Krsul and Spafford make the following assertion in [24]:

> People work within certain repeated frameworks. They use those things that they are more comfortable with or are accustomed to. Humans are creatures of habit, and habits tend to persist. That is why, for example, we have a handwriting style that is consistent during periods of our life, although the style may vary as we grow older.
>
> Likewise for programming, we can ask: which are the programming constructs that a programmer uses all the

> time? These are the habits that will be more likely entrenched, the things he consistently and constantly does and that are likely to become ingrained.

However, it is important to understand fundamental differences besides experience between the professional and student code samples. For one, student code files will have the same functional objective per assignment, whereas professional corpora will not. In addition, most student code files, per assignment, will be roughly the same file size/lines of code and will probably be much smaller than submitted professional code files. Therefore, the professional corpora will be used to answer the research question below:

How does the size of the author's profiles affect performance of the approach? Will a larger profile, or larger file size, say 2000 lines of code per author, cloud the results of the algorithms,help fine tune a more discernable identification, or have no difference?

As mentioned previously, cross-entropy is a measure of the unpredictability of a given event, given a specific (but not necessarily best) model of events and expectations [10]. More source code would seem to give a better representation of the model. It would seem the more code available to the method, the more opportunities for distinctive identifiers to manifest themselves in the corpora, therefore, one testable hypothesis is:

More lines of code per author should yield better predictive results.

As previously mentioned, a major difference between the professional corpora and the corpora of students is the purpose of the code samples. The source code corpora comprising professional code differs greatly from the student corpora because the professional code samples will not be a duplication of the same functionality, i.e. a programming assignment, but rather contributions of code pieces in a team environment

with a common goal. The computer science student corpora will have the same objective with regard to functionality. This should reduce the noise in the data set by having the same core libraries embedded in the code. For example, if the programming assignment focuses on implementation of a socket level listening device, then it can be assumed that most students will choose to incorporate libraries related to socket programming. This leads to the next research question, which is as follows:

Will the cross-entropy approach more accurately identify authors where the test bed programs have the same functionality/objective than for programs that have varying objectives? Will the algorithm be able to detect functional similarity?

In other words, will this reduction in noise help the algorithm differentiate between programmers, or because more authors are using the same constructs, restrict the performance of the algorithm because source code samples are now more similar, blurring the lines of differentiation? This will be especially true considering the nature of the student corpora.

Another way to think about this issue is to imagine constraints placed on individual handwriting so that everyone must conform to the same font, size, spacing, replication of characters, etc., and the only flexibility available for an individual characteristic is with the letters *i* and *t*. All things being equal, is there enough substantial information in the way authors dot their *i*'s and cross their *t*'s to distinguish them from other authors? For some individuals the answer is probably yes, especially if their characteristic is unique within the entire population. For example, if John always crosses his *t* slanting from bottom to top, then it would be easy to identify his handwriting among the population, with all else being equal. This reduction in noise has helped easily identify John. However, if John has no unique identifying characteristic (i.e., dots his *i*'s

like everyone else), the available avenues to express his individuality have been constrained.

Beyond functional/objective similarity of code samples, the cross-entropy approach uses a sliding window of size n. To further investigate accurate classification with respect to window size, the performance of the cross-entropy algorithm will be explored and documented in an effort to understand the optimal window size for various code types.

What is the role of window size in correct classification based on functional objective, experience, size, etc.?

Windows where n is small may not capture author identification fingerprints completely or give high marks for matches that are a result of language constraints. For example, if the window size is 13, keyword combinations such as public float x would return matches for longest length of 11, when in reality most programmers could have a number of declarations in a declaration section containing such statements as public float y, public float z.

Cross-entropy is a nonfeature/metric-oriented, distance-based approach. Additional research will be conducted on the performance of cross-entropy compared to other nonfeature approaches. More formally stated:

Is cross-entropy more or less accurate than other nonfeature/metric-based approaches when determining source code authorship, such as the N gram approach mentioned in Chapter 2, which boasts 100 percent classification accuracy [36,45]?

This problem will be explored by selecting a set of other known authorship identification approaches that will be applied to the same corpus to determine how the cross-entropy algorithm accuracy compares with the N-Gram aproach and whether or not

the N-Gram approach will be able to determine authorship identification. Some recent authorship identification techniques, such as Principal Component Analysis (PCA) [50] and LDA,, take the approach of identifying, by close inspection, a "fingerprint" or characteristic of the author, and then determining whether this fingerprint is also present in the test case. A fingerprint is created by calculating the frequencies of the most common 30, 50, or 100 words as variables and by counting them in texts or parts of texts. The rate of occurrence turns out to be rich in information about style. However, Juola makes the assertion the cross-entropy approach outperforms both PCA and LDA [10]. What is not mentioned is a comparison against the N-Gram approach discussed in [36, 46, 48].

As mentioned previously, the N-Gram byte level approach [36, 46-48], takes source code documents of known authorship, and divides them into sequences of n grams while assigning a normalized frequency to each gram. A table is then produced showing all n-grams in a document and their associated frequencies, sorted highest to lowest. This is known as the author's profile. A test document of unknown authorship is then run through the same process. The N-Gram frequency profiles from the training documents and the test documents are then compared. A threshold is then set, where the top L N-Grams are compared between documents, and the tables that have the most intersects are treated as more similar. In other words, the author who has the most number of common N-Grams shared between the author's profile and the test profile is chosen as the solution.

A direct comparison of both techniques, across all corpora, should show if cross-entropy performs better than N -Gram when applied to source code. The N-Gram

experiments will be constructed as follows, and follow the basic guidelines outlined in [36, 46, 48]:

1. Every file in each corpora will have an N-Gram table constructed using the same N-Gram processing package used in approach [36, 46, 48]. The package was written in Perl by V. Keselj [47] and can be found at the Massachusetts Institute of Technology Artificial Intelligence Laboratory website, at http://www.ai.mit.edu/courses/6.863 /Ngrams.html.
2. Three N-Gram tables will be constructed at sizes 3-grams, 4-grams, and 5-grams. The n-gram tables will be sorted by frequency of occurrence in the file. These N-Gram tables are known as the user's profile.
3. Each N-Gram table, or profile, from step 2 will be compared with all other N-Gram tables using pairwise comparison of the same N-Gram size, taking as a threshold the top 200 grams, 500 grams, and 1000 grams.
4. A candidate author will be chosen by determining which files have the highest intersection, or the number of common n-grams with the test file in question. This is the same approach taken by Frantzeskou et al. in [36, 45, 48].

It must be noted that one step is missing from the approach taken in [36, 45, 48] and these proposed steps. In the approach taken by Frantzeskou et al., multiple files by the same author were concatenated to create a profile. Because of the small number of samples in the corpora obtained from the professional and the students, all having contributed five files or fewer per author, concatenation of two files would constitute almost half of the corpora, leaving a small set to test against. Although a large cross-entropy database representing the user's profile could be constructed by concatenating multiple files, giving the cross-entropy approach a higher probability of finding a match,

it was decided not to do so because of the limited amount of test files. In addition, and more importantly, in order to remain consistent with the literary works experimental approach by Juola [10], which used pairwise comparison, the cross-entropy approach will use pairwise comparison; therefore, the N-Gram approach should probably also follow this paradigm to keep things consistent. Therefore this work proposes using only one file as the author profile.

CHAPTER IV

APPLICATION OF CROSS-ENTROPIC APPROACHES TO SOURCE CODE CORPORA: EXPERIMENT EXECUTION AND RESULTS

4.1 Experiment Executions and Results Overview

In all, the experiments in this section generated about 500,000 cross-entropy experiment executions with almost 24,500 file comparisons. The following sections describe the experiments, executions, results, and subsequent experiments. A table describing the design, parameters, logistics, research questions addressed, and attributes for the execution of an experiment will precede the discussion of each experiment. The discussion will entail results of the experiments, surprise findings, and complications or unintended consequences discovered during the experiment. A summary will discuss the accuracy of the findings of the experiment.

4.2 Cross-entropy Approach Applied to Professional Corpora

The focus of this section is to test the predictive capabilities of the cross-entropy approach on source code samples from professional programmers to determine accuracy of the cross-entropy approach when discerning authorship.

4.2.1 Experiment E1 - Can the Cross-Entropy Approach Predict Code Authorship to a Level Comparable to Literary Classifications?

For Experiment E1, the cross-entropy algorithm was applied to the 25 source code files from the professional corpora. The source code submissions contained the following properties and constraints:

1. Files submitted were not necessarily from the same program.
2. All code samples submitted comprised the author's own work containing no shared authorship within files.
3. All authors were professional programmers from the U.S. Army Engineer Research and Development Center in Vicksburg.
4. All source code documents were written in C#.Net programming language.

For the experiment, each source code sample was compared pairwise against all other files, meaning that each code file would have the opportunity to be the file of unknown authorship, matched against all others. (As mentioned in Section 3.1.3 of Chapter 3, a pairwise comparison means that each source file will be compared against all other source files, making the number of comparisons equal to $n * (n-1)$ where n is equal to the number of files in the experiment, or in this case, the corpora. The professional programming corpora are composed of 25 files; therefore, in order to compare every file against all others, a total of 600 comparisons were made (excluding comparison against itself) or 25 * (25-1) comparisons.

In addition, each pairwise cross-entropy comparison was executed 20 times. This is because cross-entropy can use a variable window size. The experiment compared the files using variable window sizes from 5 to 100 in increments of 5 for a total of 20 runs per file, which results in 12,000 runs. (In other words, each of the 25 unknown documents was compared against the other 24 documents 20 times apiece, with the first comparison of a pair of documents using a sliding window size of 5, the next iteration a window size of 10, and so on, until the window size reached 100 characters.) Obviously, the purpose here was to determine the effect of the window size on correct classification and what the optimal window size might be.

For each comparison, the mean match length was calculated and stored along with the size of the window. A sorting routine then parsed the results of the 12,000 comparisons, looking for the highest mean match length at each window size. (For a particular file designated as unknown, there were 480 runs, each focused on a window size from 5 to 100, in increments of 5, for 20 runs per comparison. The sorting routine, described in Section 3.1.4, took the top candidate author, per window size, across all comparisons, and created a results output file indicating the candidate with the highest match length determined by the cross-entropy algorithm.)

4.2.2 Discussion of Experiment E1 Results

The results of Experiment E1 were not encouraging. However, the results did provide some valuable insights for subsequent experiments and some possible limitations of the cross-entropy algorithm.

For Experiment E1, summarized in Table 4.1, of the 25 files, 13 correctly identified the author at some window size for 52 percent accuracy, with most correct classification coming at window sizes of 20 or less. These odds might first appear to be the odds of flipping a coin. While this accuracy may seem low, this is far above random assignment, considering that randomly choosing the correct author is only 1 out of 5 (there are five authors in this experiment) for a 20 percent probability. For more details, see Table 4.2.

Table 4.1 Summary of Experiment E1

Research Questions Addressed	Will the cross-entropy approaches taken in literary document classification and authorship identification fare similarly when compared with cross-entropy approaches applied to source code authorship identification? Will the cross-entropy approach more accurately identify experienced programmers when compared against less experienced programmers on programs with similar objectives. How does the size of the author's profiles affect performance of the approach? Will a larger profile, or larger file size, say 2000 lines of code per author, cloud the results of the algorithm, help fine-tune toward a more discernable identification, or have no difference?
Hypotheses	A cross-entropy applied to source code will be able to predict authorship.
Experimental Group	Computer programmers who provided source code samples for the experiment
Control Group	None
Independent Variable	None
Dependent Variable	None
Confounding Variables	Size of source code samples
Experiment Subject Population	Professional programmers at the U.S. Army Engineer Research Development Center, Information Technology Laboratory, Vicksburg, MS
Number of Subjects	5
Experiment Site	U.S. Army Engineer Research Development Center, Information Technology Laboratory, Vicksburg, MS
Experiment Method	Subjects volunteered for the experiment and signed a consent form. Subjects emailed source code samples for five source code files determined arbitrarily by the subject of at least 150 lines of code. Source code files provided could be written by the providing author only, meaning original work and excluding files created in tandem.
Experiment Preparations	The personal identifying information (name) of all subjects was stripped from the source code files and reassigned using unique identifiers for the purpose of anonymity. Source code files were divided into different directories for the parallel processing in different shell scripts.
Required Resources	Anonymized source code files. An implementation of the cross-entropy algorithm. A processing program to generate the results through pairwise comparisons at incremental window sizes. An analysis program to sift through the results identifying the most likely candidate author. A MacPro with 16GB of memory and dual quad cores. To perform the experiment, a high-end machine was desired to reduce run time.
Incentives	None

The results of Experiment E1 having a 52 percent accuracy rate was not as surprising as something the experiment uncovered. Remarkably, when the incorrect author was chosen, two files were consistently selected by the algorithm to be candidate

author about 75 percent of the time. Out of 24 possible files to choose as the best match, these two files were chosen over and over as the closest matching source code.

To clarify, there are 25 source code files. Each file had an opportunity to be the unknown author to be compared against the other 24 files, at window sizes from 5 to 100. From the results, two source code files were consistently chosen as the candidate author, for some number of *n* window size, in 19 of 25 tests runs, for a classification (or rather misclassification) rate of 76 percent. It becomes obvious from these results that something in the data set of these two files skewed the accuracy, or a limitation was associated with the algorithm and/or how it handled data sets. The question then became why. Further investigation revealed more surprising results.

The results from Experiment E1 showed a high correlation with two authors' work for unknown reasons. The algorithm consistently discerned a relationship between the two authors' source code files at all window sizes. However, the source code samples were not written for the same application/project, have totally separate functional purposes, and were not similar in size/lines of code.

Upon the visual inspection, the reasoning behind the relationship appeared apparent (although later experiments show the reasoning invalid). Although the files were totally unrelated, the code files shared some unintended commonalities. The first source code file was written with the purpose of displaying charting functions and visual graphs. There were many references to the variable name "chart," charting libraries, and the variable name "index," such as shown in Figure 4.1.

Table 4.2 Results for Experiment E1

Author and File	Correct Match File	Window Sizes
A-1	-	-
A-2	A-3	5,10,50-100
A-3	-	-
A-4	-	-
A-5	-	-
B-1	B-3	5-25
B-2	B-3	5-15
B-3	B-1,B-2	5-15
B-4	-	-
B-5	-	-
C-1	-	-
C-2	C-4	5-100
C-3	-	-
C-4	-	-
C-5	C-4	5-100
D-1	D-4	5-100
D-2	D-3	5-20
D-3	-	-
D-4	D-5	5-100
D-5	D-4	5-100
E-1	-	-
E-2	E-5	45-100
E-3	E-4	5-100
E-4	E-3	5-100
E-5	-	-

The second source code file was written with the purpose of parsing and evaluating regular expressions with looping. There were many references to the variable name “characters” and the variable name “index”, such as shown in Figure 4.2.

```
List<TokenResult> finalResult = new List<TokenResult>();
....
characters.Add("(", "LPAREN 1");
characters.Add(")", "RPAREN 2");
characters.Add("[", "LBRACK 3");
.....
public bool assignValue(string name, string index, string value, bool print)
    {
```

Figure 4.1 Code Snippet

```
ResultChart chart = new ResultChart() .
{ Index = index, K = index, Legend = type, Element = "", Data = doubleArrayToByteArray(data) };
result.Add(chart);
index++;..
```

Figure 4.2 Code Snippet

From a quick visual inspection of Figure 4.1 and 4.2, and with an understanding of how the cross-entropy algorithm functions, it becomes apparent the longest prefix of "char," appearing in the "chart" from one source code file, and "char" appearing in the word "character" from the other source code file, is shared between the two source code samples and is found in abundance (as well as the variable name "index"), thereby increasing the mean match length over the comparison and leading to a misclassification.

4.2.3 Experiment E2 – Cross Entropy with standard deviation

One way to is to eliminate multiple occurrences of "char" from being added into the mean match length computation is by finding only the longest prefixes by computing the mean match length a priori, along with the standard deviation, and only allowing match lengths above of one SD of the into a subsequent match length computation. By doing so, the algorithm will only examine the outlying nuances that are the longest matches between two sets of code. It would seem logical that the longer the matches, the

lower the probability of randomly finding a match between two code files, and the higher probability the code was intentionally written by the same author and shared between source code samples.

Table 4.3 Summary of Experiment E2

Research Questions Addressed	Can a cross-entropy approach be used to predict source code authorship? What is the effect of window size on accuracy?
Hypothesis	A cross-entropy approach applied to source code will be able to predict authorship.
Experimental Group	Computer programmers who provided source code samples for the experiment
Control Group	None
Independent Variable	None
Dependent Variable	None
Confounding Variables	Size of source code samples
Experiment Subject Population	Professional programmers at the U.S. Army Engineer Research Development Center, Information Technology Laboratory, Vicksburg, MS
Number of Subjects	5
Experiment Site	U.S. Army Engineer Research Development Center, Information Technology Laboratory, Vicksburg, MS
Experiment Method	Subjects volunteered for the experiment and signed a consent form. Subjects emailed source code samples for five source code files determined arbitrarily by the subject of at least 150 lines of code. Source code files provided could be written by the providing author only , meaning original work, and excluding files created in tandem.
Experiment Preparations	All personal identifying information (name) of the subjects was stripped from the source code files and reassigned using unique identifiers for the purpose of anonymity. Source code files were divided into different directories for parallel processing in different shell scripts.
Required Resources	Anonymized source code files An implementation of the cross-entropy algorithm A processing program to generate the results through pairwise comparisons at incremental window sizes An analysis program to sift through the results identifying the most likely candidate author A MacPro with 16GB of memory and dual quad cores. To perform the experiment, a high-end machine was desired to reduce run time.
Incentives	None

4.2.4 Discussion of Experiment E2 Results

The results of experiment E2 were very similar to E1. Limiting the focus to longer match lengths, those at least one standard deviation longer than the average match length, did not significantly change the results.

As in experiment E1, of the 25 files in experiment E2, 13 correctly identified the author at some window size for 52% accuracy, with most correct classification coming at window sizes of 20 or less. In addition, when the incorrect author was chosen, the same two files were consistently selected by the algorithm to be the candidate author in 21 of the 25 files, which is an increase from experiment E1.

To clarify, there are 25 source code files and each file gets an opportunity to be the unknown author to be compared against the other 24 files, at window sizes from 5 to 100. From the results, two source code files were consistently chosen as the candidate author, for some number of n window size, in 21 of 25 tests runs, for a classification (or rather misclassification) rate of 84%.

There must be some reason these two files are showing high classification rates. The only obvious feature shared between the two files in question, are their file sizes. One file is the largest, which has the highest number of misclassifications, while the other is the smallest. It seems the cross entropy algorithm has an accuracy bias related to file size distribution when comparing files within the corpora. One way to test this theory is to eliminate the files in question from subsequent experiments and analyze the results.

4.2.5 Experimental E3 – Cross-entropy Approach with Respect to File Sizes

Because of the misclassifications discovered in Experiment E1 and the insignificance of standard deviation applied in Experiment E2, it was decided to reperform the experiments in Experiment E1, sans the files in question to boost accuracy. Interestingly, the two highly misclassified files in question (henceforth known as F1 for the largest and F2 for the smallest file) happened to be the largest and smallest within the corpora, which leads to a theory.

Empirically, there seems to be a relationship between cross-entropy correctly classifying authorship and the distribution of file size in an experiment. The average file size for the 25 files, from the five different authors, ranged from one file with a size of 176 kilobytes (file F1), to a file with a size of 4 kilobytes (file F2). The average file size for all 25 files is 44.16 kilobytes. As mentioned previously, and in Experiment E1, F1 and F2 were chosen as the candidate author 76 percent of the time for some window size *n,* even though this was a misclassification. The larger file, F1, is four times larger than the file size average, and file F2, the smallest, is ten times less than the average.

This particular experiment also helps address one of the research questions posed in Chapter 3, in particular:

How does the size of the author's profiles affect performance of the approach? Will a larger profile, say 2000 lines of code per author, cloud the results of the algorithm,help fine-tune toward a more discernable identification, or have no difference?

Obviously, there seems to be a relationship between size of files and classification accuracy within the corpora that limits the performance of the cross-entropy. Intuitively, the author believes the reason behind the misclassifications for the larger file is because the larger file has enough code to match most of the other source code samples. It is somewhat of a superset that can be thought of as a dictionary, which contains many chances for matches when window sliding. For example, imagine looking up every word in a document in a dictionary and scoring positively for finding a reference. It could be assumed that the person that wrote the dictionary and the document was the same author because of the common intersection of the words (at least from the perspective of the cross-entropy algorithm). However, if the dictionary is compared using the document as a reference, it would produce a very low score because the dictionary has far more words

not contained in the document, thereby scoring negatively for a lower correlation. (In fact, what is interesting is that when F1, by far the largest file, is the unknown document compared with the other source code files using cross-entropy, the most common classification, although incorrectly classified, is the next largest file within the corpora at 143 kilobytes.)

As for the misclassification regarding the smallest document F2, the author intuitively believes F2 scored high mean match length values because it contains only the key words found in almost all of the samples, such as using statements (these are similar to import statements in C, and many are autogenerated by development tools). For example, see Figure 4.3.

```
using System;.
using System.Collections.Generic;.
using System.Linq;.
using System.Net;.
using System.Windows;.
using System.Windows.Controls;.
using System.Windows.Documents;.
using System.Windows.Input;.
using System.Windows.Media;.
using System.Windows.Media.Animation;.
using System.Windows.Shapes;.
using SWWRP_KMDS.Web.DomainServices;
using SWWRP_KMDS.Web.EntityModels;.
```

Figure 4.3 Snippet from Smallest Source Code File in Professional Corpora.

This smallest file (F2) is around 50 lines long and 13 of those lines are using statements. It is believed that the issue with this particular misclassification is that the algorithm never had an opportunity to lower the mean match length because it started off scoring high correlations and finished processing quickly because of small file size, all before the negative scores (or no matches) could lower the mean average.

What does this mean? The supporting evidence seems to suggest that the cross-entropy method is size dependent. Files within a corpus may need to be distributed evenly for more accurate results. Experiment E3 will help validate this theory and try to boost accuracy to around levels seen in literary analysis [10].

Table 4.4 Summary of Experiment E3

Research Questions Addressed	Does the removal of files with sizes significantly larger or smaller than the average file size for the corpora samples have a positive or negative effect on authorship prediction? How does the size of the author's profiles affect performance of the approach? Will a larger profile, say 2000 lines of code per author, cloud the results of the algorithm, help fine-tune toward a more discernable identification, or have no difference? Will cross-entropy approaches taken in literary document classification and authorship identification fare similarly when compared with cross-entropy approaches applied to source code authorship identification?
Hypotheses	Having candidate files of roughly the same size distribution will increase accuracy of the cross-entropy approach.
Experimental Group	Computer programmers who provided source code samples for the experiment
Control Group	None
Independent Variable	None
Dependent Variable	None
Confounding Variables	Size of source code samples
Experiment Subject Population	Professional programmers at the U.S. Army Engineer Research Development Center, Information Technology Laboratory, Vicksburg, MS
Number of Subjects	5
Experiment Site	U.S. Army Engineer Research Development Center, Information Technology Laboratory, Vicksburg, MS
Experiment Method	Subjects volunteered for the experiment and signed a consent form. Subjects emailed source code samples for five source code files determined arbitrarily by the subject of at least 150 lines of code. Source code files provided could be written only by the providing author, meaning original work, and excluding files created in tandem.
Experiment Preparations	Personal identifying information (name) of all subjects was stripped from the source code files and reassigned using unique identifiers for the purpose of anonymity. Source code files were divided into different directories for parallel processing in different shell scripts.
Required Resources	Anonymized source code files An implementation of the cross-entropy algorithm A processing program to generate the results through pairwise comparisons at incremental window sizes An analysis program to sift through the results identifying the most likely candidate author A MacPro with 16GB of memory and dual quad cores. To perform the experiment, a high-end machine was desired to reduce run time.
Incentives	None

4.2.6 Discussion of Experiment E3 Results

The results of Experiment E3 were much more encouraging than the results from Experiment E1. Of the 23 files, 18 correctly identified the author at some window size for almost 80 percent accuracy. (Note: There are 23, not 25, files because the largest and smallest files that were causing problems in Experiment E1 were removed.) See Table 4.5 and 4.6 for more information.

What was surprising is that of the five runs that misclassified at all window sizes, the same candidate file showed up in all. Interestingly, the file was also the smallest in the reduced corpora, with a file size of 8 kilobytes, further lending credence to the idea there is a correlation between file size and distribution with respect to average corpora file size and classification accuracy.

The results from this experiment show that cross-entropy can perform with accuracy similar to literary classification results [10]; however limitations of the candidate sizes must be taken into account in order to ensure proper file size distribution to reduce limitations of the algorithm.

Table 4.5 Results for Experiment E3

Author and File	Correct Match File	Window Sizes
A-1	A-4	10
A-2	A-3	5-100
A-3	A-1, A-2	5-15, 55-100
A-4	-	-
A-5	-	-
B-1	B-3	5-100
B-2	B-3	5-100
B-3	B-2, B-1	10-25, 5, 30-100
B-4	B-1, B-2	25-100, 5-10
B-5	B-3	5
C-1	C-3	5-100
C-2	-	-
C-3	C-1	5-100
C-4 Removed	Removed	Removed
C-5	-	-
D-1	D-4	5-100
D-2	D-3	5-100
D-3	D-4	10-100
D-4	D-5	5-100
D-5	D-3	5-100
E-1	E-3	5-15, 80-100
E-2	-	-
E-3	E-4	5-100
E-4	E-3	5-100
E-5 Removed	Removed	Removed

Table 4.6 Correct Authorship Classifications per Window Size for Experiment E3

Window Size	**Number of Correct Classifications**
5	16
10	17
15	15
20	13
25	14
30	14
35	14
40	14
45	14
50	14
55	15
60	15
65	15
70	15
75	15
80	16
85	16
90	16
95	16
100	16

4.2.7 Professional Corpora Discussion

The professional corpora did find encouraging results in Experiment E3, pushing classification of the 23 files in the corpora up to 80 percent. However, some limitations of the approach were discovered that must be considered when the corpora test bed is set up, mainly similar file size distributions. This is the end of the professional corpora experiments with respect to cross-entropy. However, there are still many more corpora to be tested, all of which have smaller file sizes and similar file distributions. The professional corpora will be reexamined later in this chapter when the N-Gram technique is applied, thereby providing a measuring stick for cross-entropy performance.

4.3 Cross-Entropy Approach Applied to Student Corpora and Smaller File Sizes

This section describes a number of experiments conducted using three different corpora provided by professors or instructors from Mississippi State University. All corpora are studied independently because of the diverse programming languages implemented in each course.

This section will try to address the following research questions discussed in Chapter 3:

The cross-entropy approach will more accurately identify experienced programmers when compared against less experienced programmers on programs with similar objectives.

What is the role of window size with respect to correct classification based on objective, experience, size, etc.?

Will the cross-entropy approach more accurately identify authors where the test bed programs have the same functionality/objective, versus programs that have varying objectives? Will the algorithm be able to detect functional similarity?

4.3.1 Student Corpora Experiments for Course 1

The first of the student corpora experiments used the Course 1 corpora, referenced in Section 3.1.2.2. Course 1 was composed primarily of electrical engineering students. The purpose of the course was to provide an introduction to software programming using the C++ programming language. The Course 1 corpora comprised 81 source files from seventeen different student authors for 6480 cross-entropy experiment runs at 20 window sizes apiece, totaling 129,600 comparisons. All programming assignments were relatively short by programming standards, with the largest files at just over 100 lines of code.

4.3.1.1 Experiment E4– Course 1 Experiment within Assignment

For Experiment E4, summarized in Table 4.7, all files were compared to all other files within the Course 1 corpora, regardless of functionality of course assignment. This was the first experiment, but certainly not the last, where the authors of the code sought the same functional objective within their source code assignments. The focus of these experiments, when compared to the professional corpora experiments, was to examine how the cross-entropy algorithm performs on the smaller sample sizes, what is the affect of same functional/objective between codes and classification, and to gauge how accuracy may increase as the students gain experience over the semester.

4.3.1.2 Discussion of Experiment E4 Results

Preliminary analysis of the results proved discouraging, with only 14 source code files out of 81 finding the correct author. However, a closer visual analysis of the results showed a high level of code sharing between students on assignments, as shown in Figure 4.4 and 4.5.

Table 4.7 Summary of Experiment E4

Experiment ID	E4
Research Questions Addressed	Does the cross-entropy method provide similar accuracy results to those in Experiment E3, when applied to a student corpora whose file size samples are much smaller? Will cross-entropy accuracy increase as the semester progresses and programmers become more experienced? Will the cross-entropy approach more accurately identify authors where the test bed programs have the same functionality/objective, versus programs that have varying objectives? Will the algorithm be able to detect functional similarity?
Hypotheses	The cross-entropy approach will more accurately identify experienced programmers when compared against less experienced programmers on programs with similar objectives. Accuracy will increase as programmers gain more experience. The cross-entropy approach will less accurately identify authorship when the source files have the same functional/objective approach.
Experimental Group	Computer science students from Mississippi State University who provided source code samples for the experiment
Control Group	None
Independent Variable	None
Dependent Variable	None
Confounding Variables	Size of source code samples
Experiment Subject Population	Computer science students from Mississippi State University who provided source code samples for the experiment
Number of Subjects	17
Experiment Site	ERDC, ITL, USACE, Vicksburg, MS
Experiment Method	Subjects volunteered for the experiment and signed a consent form. Subjects emailed source code samples for five source code files determined arbitrarily by the subject of at least 150 lines of code. Source code files provided could be written by the providing author only, meaning original work, and excluding files created in tandem.
Experiment Preparations	All personal identifying information (name) of the subjects was stripped from the source code files and reassigned using unique identifiers for the purpose of anonymity. Source code files were divided into different directories for parallel processing in different shell scripts.
Required Resources	Anonymized source code files An implementation of the cross-entropy algorithm A processing program to generate the results through pairwise comparisons at incremental window sizes An analysis program to sift through the results identifying the most likely candidate author A MacPro with 16GB of memory and dual quad cores. To perform the experiment, a high-end machine was desired to reduce run time
Incentives	None

While plagiarism is not the focus of this research, in a controlled setting such as a classroom, where multiple programmers have the same objectives (getting a good grade), the same goal (namely a programming assignment), and with a subset of students lacking in programming skills with deadlines quickly approaching, shortcuts will be taken to ensure completion.

Still, the case can be made that the purpose of this research is to identify authors of unknown source code. The experiment did show that a single author, in this case a student, more than likely wrote and shared the programs or "ideas" in question in Figures 4.4 and 4.5.

The author does not believe the cross-entropy approach failed in the endeavor of determining authorship. The algorithm did determine the authorship. The issue is that one of the source codes is an almost identical replication of the original source, submitted as an independent effort. The confounding issue for the algorithm is the sharing of an author's work with other students. The experiment relies on the assumption that source code files submitted are from an author's own hand.

Still, while the lack of clear code ownership in the corpora proved a hindrance, the algorithm was very effective at discerning assignment similarity, or shared functional objective, much like the literary equivalent of discerning document topic. This is to be expected considering cross-entropy is a distance-measurement based algorithm, and the same functional/objective within codes should have a lower distance. In fact, the algorithm selected the correct assignment, just with a different author (obviously since there was copying between assignments and a source code cannot be compared to itself, it will find the closest match, in this case a copy of the assignment by another student) in 67 out of 81 source code samples for an assignment classification rate of 82 percent.

What is more interesting is that out of the 14 source code samples the algorithm classified as not the same assignment, 14 of the 14 choose the correct author. In addition, all 14 assignments in question were the last two assignments in the semester, indicating that these assignments would have occurred later in the year after the students gained more programming experience.

cse1233-6-6

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

float calculateCharges(float h);

main()

{
        float car[8] = {0};
        float charge[8] = {0};
        float totalHours = 0.0;
        float totalCharges = 0.0;
        int i = 0;

        for (i=0; i<8; i++)
        {
        printf("Enter the hours for car %d: ", i+1);
        scanf("%f", &car[i]);
        totalHours = totalHours + car[i];
        charge[i] = calculateCharges(car[i]);
        totalCharges = totalCharges + charge[i];
        }

        printf("\n");
        printf("Car\t\tHours\t\tCharge\n");
        printf("1\t\t%.1f\t\t%.2f\n", car[0], charge[0]);
        printf("2\t\t%.1f\t\t%.2f\n", car[1], charge[1]);
        printf("3\t\t%.1f\t\t%.2f\n", car[2], charge[2]);
        printf("4\t\t%.1f\t\t%.2f\n", car[3], charge[3]);
        printf("5\t\t%.1f\t\t%.2f\n", car[4], charge[4]);
        printf("6\t\t%.1f\t\t%.2f\n", car[5], charge[5]);
        printf("7\t\t%.1f\t\t%.2f\n", car[6], charge[6]);
        printf("8\t\t%.1f\t\t%.2f\n", car[7], charge[7]);
        printf("TOTAL\t\t%.1f\t\t%.2f\n", totalHours, totalCharges);
}

float calculateCharges(float h)
        {
                const float MINIMUM_CHARGE = 2.0;
                const float MAXIMUM_CHARGE = 10.00;


                if (h<=3)
                        return MINIMUM_CHARGE;


                h = h - 3;


                float additionalAmount = (h) * 0.50;

                float charge = MINIMUM_CHARGE + additionalAmount;

                if (charge > MAXIMUM_CHARGE)
                        return MAXIMUM_CHARGE;
                else
                        return charge;
        }
```

cse1233-6-1

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

float calculateCharges (float h);

main()
{
        float car[8] = {0};
        float charge[8] = {0};
        float totalHours = 0.0;
        float totalCharges = 0.0;
        int i =0;

        for (i = 0; i < 8; i++)
        {
        printf("Enter the hours for car %d: ", i + 1);
        scanf ("%f", &car[i]);
        totalHours = totalHours + car[i];
        charge[i] = calculateCharges (car[i]);
        totalCharges = totalCharges + charge[i];
        }

        printf ("\n");
        printf ("Car\t\t Hours\t\t Charge\n");

        for (i = 0; i < 8; i++)
        {
             printf ("%d\t\t %.1f\t\t %.2f\n", i + 1, car[i], charge[i]);
        }
        printf ("TOTAL\t\t %.1f\t\t %.2f\n", totalHours, totalCharges);
}

float calculateCharges (float h)
        {
        const float MINIMUM_CHARGE = 2.0;
        const float MAXIMUM_CHARGE = 10.00;

        if (h <= 3)
            return MINIMUM_CHARGE;

        h = h - 3;
        float additionalAmount = (h) * 0.50;
        float charge = MINIMUM_CHARGE + additionalAmount;

        if (charge > MAXIMUM_CHARGE)
             return MAXIMUM_CHARGE;
        else
             return charge;
        }
```

Figure 4.4 Similarities Between Two Students' Source Code Samples

Does this mean these authors started developing an individual style discernable by the cross-entropy algorithm? This is difficult to say. What is noticeable in visual inspection of correctly classified authorship source files is the reuse for certain sections of the code from one assignment to the next. This could mean these students had the beginnings of developing a toolbox to solve certain problems.

Still, while the lack of clear code ownership in the corpora proved a hindrance, the algorithm was very effective at discerning assignment similarity, or shared functional objective, much like the literary equivalent of discerning document topic. This is to be expected considering cross-entropy is a distance-measurement based algorithm, and the same functional/objective within codes should have a lower distance. In fact, the algorithm selected the correct assignment, just with a different author (obviously since there was copying between assignments and a source code cannot be compared to itself, it will find the closest match, in this case a copy of the assignment by another student) in 67 out of 81 source code samples for an assignment classification rate of 82 percent.

What is more interesting is that out of the 14 source code samples the algorithm classified as not the same assignment, 14 of the 14 choose the correct author. In addition, all 14 assignments in question were the last two assignments in the semester, indicating that these assignments would have occurred later in the year after the students gained more programming experience.

cse1233-6-12

```
#include <math.h>

#include <stdio.h>

float calculateCharges(float);
const int NUMBER_OF_CARS = 8;

main()

{
int i = 0;
int cars[NUMBER_OF_CARS];
float charges[NUMBER_OF_CARS];
float hours[NUMBER_OF_CARS];
float charge = 0.0;
float totalHours = 0.0;
float totalCharges = 0.0;


 for (i=0; i<(NUMBER_OF_CARS); i++)
    {
        cars [i] = i+1;
        printf("\nEnter the hours for car %d:\n ", cars[i]);
        scanf("%f", &hours[i]);
        totalHours=totalHours + hours[i];
        charges[i] = calculateCharges(hours[i]);
        totalCharges= totalCharges + charges[i];
    }

printf("\nCar\tHours\tCharge\n");
 for (i=0; i<(NUMBER_OF_CARS); i++)
    {
        printf("%d\t%.1f\t%.2f\n", cars[i], hours[i], charges[i]);
    }
printf("TOTAL\t%.1f\t%.2f\n", totalHours, totalCharges);
}

float calculateCharges(float h)
{
         const float MINIMUM_CHARGE = 2.00;
         const float MAXIMUM_CHARGE = 10.00;
if (h <=3)
 return MINIMUM_CHARGE;

h=h-3;

float additionalAmount= ceil(h)*0.50;
float charge = MINIMUM_CHARGE + additionalAmount;

if(charge > MAXIMUM_CHARGE)
return MAXIMUM_CHARGE;
else
return charge;

}
```

cse1233-6-14

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

float calculateCharges(float);
const int NUMBER_OF_CARS = 8;

main()
{
 int cars[NUMBER_OF_CARS];
 float charges[NUMBER_OF_CARS];
 float hours[NUMBER_OF_CARS];
 int i = 0;
 float totalHours = 0.0;
 float totalCharges = 0.0;
 float charge = 0.0;

 for (i = 0; i < (NUMBER_OF_CARS); i++)
  {
   cars[i] = i + 1;
   printf("Enter the hours for car %d:\n", cars[i]);
   scanf("%f", &hours[i]);
   totalHours = totalHours + hours[i];
   charges[i] = calculateCharges(hours[i]);
   totalCharges = totalCharges + charges[i];
  }

 printf("\nCar\tHours\tCharge\n");
 for (i=0; i < (NUMBER_OF_CARS); i++)
  {
   printf("%d\t%.1f\t %.2f\n", cars[i], hours[i], charges[i]);
  }
   printf("TOTAL\t%.1f\t%.2f\n", totalHours, totalCharges);
}

float calculateCharges(float h)
{
 const float MINIMUM_CHARGE = 2.00;
 const float MAXIMUM_CHARGE = 10.00;

 if (h <= 3)
   return MINIMUM_CHARGE;

 h = h - 3;

 float additionalAmount = ceil(h)*0.50;

 float charge = MINIMUM_CHARGE + additionalAmount;

 if (charge > MAXIMUM_CHARGE)
   return MAXIMUM_CHARGE;
 else
   return charge;
}
```

Figure 4.5 Similarities Between Two Students' Source Code Samples

Does this mean these authors started developing an individual style discernable by the cross-entropy algorithm? This is difficult to say. What is noticeable in visual inspection of correctly classified authorship source files is the reuse for certain sections of the code from one assignment to the next. This could mean these students had the beginnings of developing a toolbox to solve certain problems.

Obviously, to increase cross-entropy accuracy, an effort needs to be made to reduce the effects of source code copying between students with regard to classification. One attempt could be to limit comparison of source files within an assignment, which will be implemented for the next experiment.

4.3.1.3 Experiment E5 – Course 1 Experiment not within Assignment

As mentioned in the previous section, assignment similarity dominated classifications because of the copying between students and same functional/objective comparisons. Experiment E5 (Table 4.8) aimed to increase authorship identification accuracy by eliminating the possibility of file comparisons within assignment. (In other words, a source code of unknown authorship, written for a particular homework assignment, will not be compared with source files from other students written for that particular assignment. In reality, this makes sense because a student can submit only one work per assignment; therefore, there is no reason to compare their work to the other code samples in the same assignment because it is impossible for the cross-entropy algorithm to match the author to himself for the assignment.)

Table 4.8 Summary of Experiment E5

Research Questions Addressed	Does the cross-entropy method accuracy increase when comparisons within assignments are eliminated? Will the cross-entropy approach more accurately identify authors where the test bed programs have the same functionality/objective, versus programs that have varying objectives?
Hypothesis	Correct authorship classification will increase from Experiment E4, once within assignment is eliminated.
Experimental Group	Computer science students from Mississippi State University who provided source code samples for the experiment
Control Group	None
Independent Variable	None
Dependent Variable	None
Confounding Variables	Size of source code samples
Experiment Subject Population	Computer science students from Mississippi State University who provided source code samples for the experiment
Number of Subjects	17
Experiment Site	ERDC, ITL, USACE, Vicksburg, MS
Experiment Method	Subjects volunteered for the experiment and signed a consent form. Subjects emailed source code samples for five source code files determined arbitrarily by the subject of at least 150 lines of code. Source code files provided could be written by the providing author only, meaning original work, and excluding files created in tandem.
Experiment Preparations	All personal identifying information (name) for subjects was stripped from the source code files and reassigned using unique identifiers for the purpose of anonymity. Source code files were divided into different directories for parallel processing in different shell scripts. **A subset of the results, comparisons of files within assignment, will be removed from the analysis.**
Required Resources	Anonymized source code files An implementation of the cross-entropy algorithm A processing program to generate the results through pairwise comparisons at incremental window sizes An analysis program to sift through the results identifying the most likely candidate author A MacPro with 16GB of memory and dual quad cores. To perform the experiment, a high-end machine was desired to reduce run time.
Incentives	None

4.3.1.4 Discussion of Experiment E5 Results

When within-assignment comparison is eliminated, authorship accuracy increases by over 115 percent, from 13 correct authorship determinations, to 28 correct authorship determinations. However, when compared to the total number of comparisons, the

correct classification rate is quite low, correctly assigning 28 out of 81 assignments for about 35 percent.

Once again, as in Experiment E4, what are encouraging are the results for the last two assignments. The last two assignments, assignments 5 and 6, comprise 31 out of the 81 files in the corpora. Out of the 28 correct classifications, 18 involved correct classifications from these two assignments, where 18 out of 31 is equal to 58 percent. Even more encouraging was assignment 6. Assignment 6 contained 15 files. Out of the 15, the algorithm classified 10 correctly, for a 67 percent classification rate.

What is unclear is whether the students' programming skills were beginning to evolve into an individual toolbox representing their style, or whether the increasing size of the assignments over the semester contributed to a more complete representation of the author's signature thereby improving the results of the algorithm. The author's guess would be a combination of the two. Figures 4.6 and 4.7 show increases in accuracy over assignments and depict file size growth over assignments, respectively, as the exercises become more in-depth. The anomaly here is Assignment 4, which was an addendum to Assignment 2. Out of the 17 source code files for Assignment 4, all 17 choose authorship from Assignment 2 as the closest match, at some window size, for 100 percent, with only a few correctly assigning the author of assignment number 2.

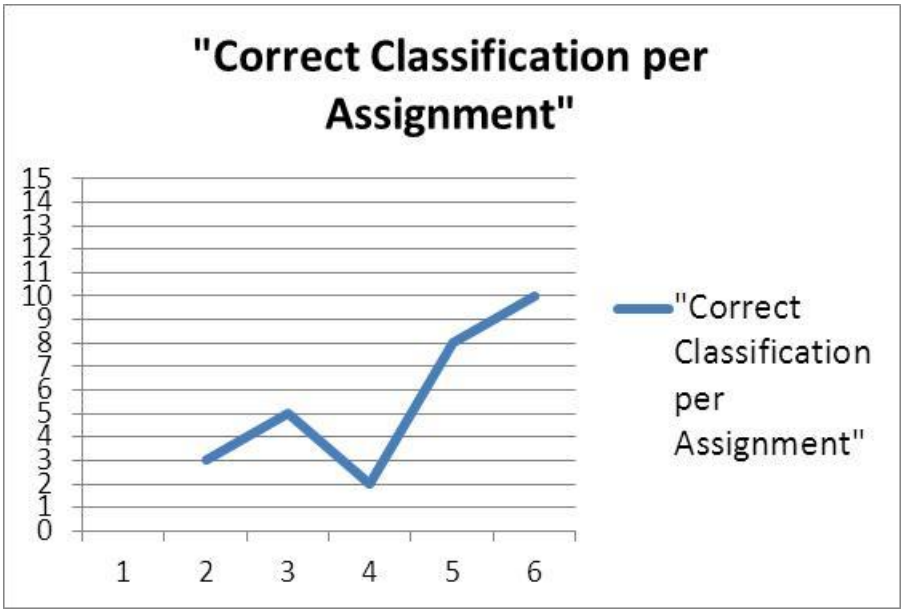

Figure 4.6 Accuracy cross-entropy increasing with assignment.

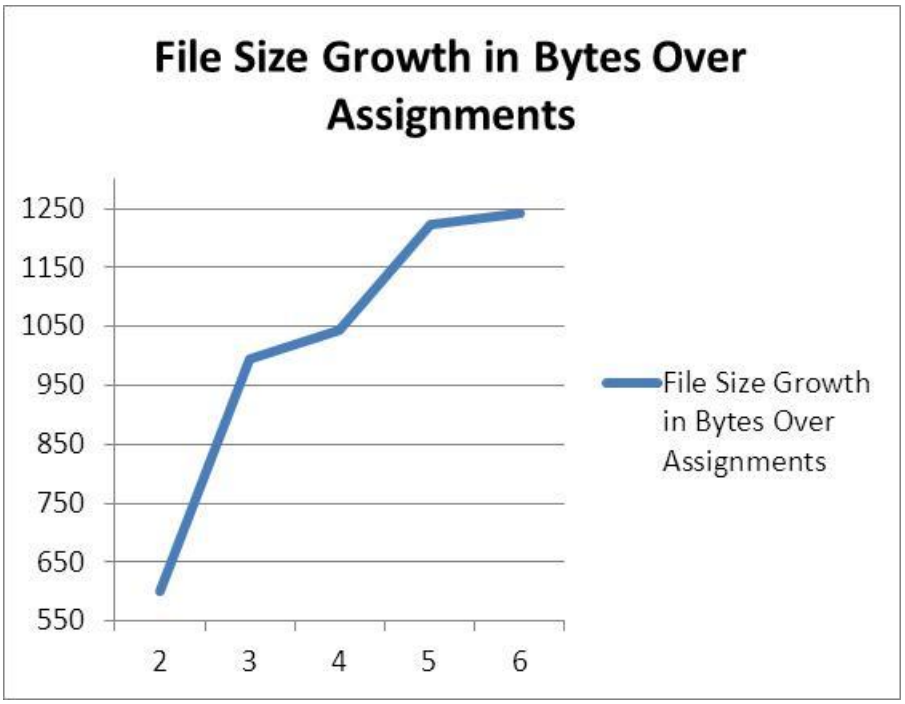

Figure 4.7 File size growth per assignment.

4.3.2 Student Corpora Experiments for Course 2

The second set of student experiments focuses on the corpora from Course 2, described in Section 3.1.2.2. Course 2 is composed primarily of computer science students, 18 of whom submitted code to this experiment, where the purpose of the course is to provide an introduction to software programming using the Java programming language. There are a total of 87 source files. Once again, using pairwise comparison, this leads to a total of 7482 cross-entropy experiment runs, computed for 20 different window sizes per run, giving a total of 149,640 comparisons.

4.3.2.1 Experiment E6– Course 2 Experiment within Assignment

For Experiment E6 (Table 4.9), all files were compared with all other files within the Course 2 corpora, regardless of functionality or course assignment. The goal of this experiment was to address the same research questions explored in Experiment E4. The purpose was to provide another data set to verify results from Experiment E4, or identify patterns not previously uncovered.

4.3.2.2 Discussion of Experiment E6 Results

Just like the results in Experiment E4, functional objective of the assignments dominated the results. The algorithm only identified 20 authors out of the 87 files for a disappointing 23 percent. Once again, copying or "idea" sharing between students (Figure 4.8) ruled where 82 times the algorithm chose a candidate within the same assignment, at some window size, for an incredibly high 94 percent selection, rather than the correct author in a different assignment.

Table 4.9 Summary of Experiment E6

Experiment ID	E6
Research Questions Addressed	Does the cross-entropy method provide similar accuracy results to those in Experiments E3 and E5, when applied to a student corpora whose file size samples are much smaller? Will cross-entropy accuracy increase as the semester progresses and programmers become more experienced? Will the cross-entropy approach more accurately identify authors where the test bed programs have the same functionality/objective, versus programs that have varying objectives? Will the algorithm be able to detect functional similarity?
Hypotheses	The cross-entropy approach will more accurately identify experienced programmers when compared against less experienced programmers on programs with similar objectives. Accuracy will increase as programmers gain more experience. The cross-entropy approach will less accurately identify authorship when the source files have the same functional/objective approach.
Experimental Group	Computer science students from Mississippi State University who provided source code samples for the experiment
Control Group	None
Independent Variable	None
Dependent Variable	None
Confounding Variables	Size of source code samples
Experiment Subject Population	Computer science students from Mississippi State University who provided source code samples for the experiment
Number of Subjects	18
Experiment Site	ERDC, ITL, USACE, Vicksburg, MS
Experiment Method	Subjects volunteered for the experiment and signed a consent form. Subjects emailed source code samples for five source code files determined arbitrarily by the subject of at least 150 lines of code
Experiment Method (Continued)	Source code files provided could be written by the providing author only, meaning original work, and excluding files created in tandem
Experiment Preparations	All personal identifying information (name) of subjects was stripped from the source code files and reassigned using unique identifiers for the purpose of anonymity. Source code files were divided into different directories for parallel processing in different shell scripts.
Required Resources	Anonymized source code files An implementation of the cross-entropy algorithm A processing program to generate the results through pairwise comparisons at incremental window sizes An analysis program to sift through the results identifying the most likely candidate author A MacPro with 16GB of memory and dual quad cores. To perform the experiment, a high-end machine was desired to reduce run time.
Incentives	None

Interestingly though, the five files that were not matched to the same assignment at some window size were matched to the correct author in a different assignment at every window size. Also, correct authorship was determined only within the last two assignments of the semester, Assignments 4 and 5, further lending credence to the idea that distinguishable styles are evolving as the student gains experience. In fact, Assignments 4 and 5 constitute 33 files. This gives an identification rate of 20 out of 33 files for a rate of 60 percent success within this subcorpus. (It should be noted that a confounding factor questioning the validity of the experiment could be that Assignment 5 appears to be a translation of Assignment 4, where the objective is the conversion of Assignment 4 from a console application to a web servlet application.)

Still, a visual inspection of the code and the cross-entropy scores support the idea that students are doing their own work over the final assignments, are reusing their own code, and developing their own mechanisms for tackling the assignment. What is needed is a comparison outside the assignment for the Course 2 corpora, such as in Experiment E5.

CSE3324-02-03-12

```
public class Simulator
{
    private Thread t1 = new Thread();
    private Thread t2 = new Thread();
    private Thread t3 = new Thread();
    private Thread t4 = new Thread();

    public Simulator()
    {
        TrafficLight Light = new TrafficLight();
        RoadRunnable r1 = new RoadRunnable(1, Light);
        RoadRunnable r2 = new RoadRunnable(2, Light);
        RoadRunnable r3 = new RoadRunnable(3, Light);
        RoadRunnable r4 = new RoadRunnable(4, Light);

        for (int i = 0; i < 10; i++)
        {
            r1.add("Maximum");
            r2.add("Maximum");
            r3.add("Maximum");
            r4.add("Maximum");
        }
         t1 = new Thread(r1);
         t2 = new Thread(r2);
         t3 = new Thread(r3);
         t4 = new Thread(r4);
    }
    public static void main(String[] args)
    {

        Simulator s = new Simulator();
         s.go();

    }

    private void go()
    {
     t1.start();
     t2.start();
     t3.start();
     t4.start();
    }
```

CSE3324-02-03-14

```
public class Simulator {

    public static void main(String[] args)
    {
        Simulator s = new Simulator();
        s.go();
    }

    public Simulator()
    {
        RoadRunnable r1 = new RoadRunnable(1, tLight);
        RoadRunnable r2 = new RoadRunnable(2, tLight);
        RoadRunnable r3 = new RoadRunnable(3, tLight);
        RoadRunnable r4 = new RoadRunnable(4, tLight);

        for(int i=0; i!=10; i++)
            r1.add("car");
        for(int i=0; i!=10; i++)
            r2.add("car");
        for(int i=0; i!=10; i++)
            r3.add("car");
        for(int i=0; i!=10; i++)
            r4.add("car");

         t1 = new Thread(r1);
         t2 = new Thread(r2);
         t3 = new Thread(r3);
         t4 = new Thread(r4);
    }

    public void go()
    {
        t1.start();
        t2.start();
        t3.start();
        t4.start();
    }

    private TrafficLight tLight = new TrafficLight();
    private Thread t1;
    private Thread t2;
    private Thread t3;
    private Thread t4;
}
```

Figure 4.8 Two Very Similar Source Code Samples from Course 2 Corpora.

The Algorithm Scored These Source Code Samples AS 93 Percent Similar at Window Size 5.

4.3.2.3 Experiment E7 – Course 2 Experiment not within Assignment

As mentioned in the previous section, and also confirmed in Experiment E4, assignment similarity dominated classifications because of the copying between students. This experiment (Table 4.10) aims to increase authorship identification accuracy by eliminating the possibility of file comparisons within assignment and helps validate the results produced in Experiment E5.

Table 4.10 Summary of Experiment E7

Experiment ID	E7
Research Questions Addressed	Does the cross-entropy method accuracy increase when within-assignment comparisons are eliminated? Will the cross-entropy approach more accurately identify authors where the test bed programs have the same functionality/objective, versus programs that have varying objectives?
Hypotheses	Correct authorship classification will increase from Experiment E7, once within assignment comparisons are eliminated.
Experimental Group	Computer science students from Mississippi State University who provided source code samples for the experiment
Control Group	None
Independent Variable	None
Dependent Variable	None
Confounding Variables	Size of source code samples
Experiment Subject Population	Computer science students from Mississippi State University who provided source code samples for the experiment
Number of Subjects	18
Experiment Site	ERDC, ITL, USACE, Vicksburg, MS
Experiment Method	Subjects volunteered for the experiment and signed a consent form. Subjects emailed source code samples for five source code files determined arbitrarily by the subject of at least 150 lines of code. Source code files provided could be written by the providing author only, meaning original work, excluding files created in tandem.
Experiment Preparations	All personal identifying information (name) for subjects was stripped from the source code files and reassigned using unique identifiers for the purpose of anonymity. Source code files were divided into different directories for parallel processing in different shell scripts. **A subset of the results, comparisons of files within assignment, will be removed from the analysis.**
Required Resources	Anonymized source code files An implementation of the cross-entropy algorithm A processing program to generate the results through pairwise comparisons at incremental window sizes An analysis program to sift through the results identifying the most likely candidate author A MacPro with 16GB of memory and dual quad cores. To perform the experiment, a high-end machine was desired to reduce run time.
Incentives	None

Again, this makes sense because a student can submit only one work per assignment; therefore there is no reason to compare their work to the other code samples in the same assignment because it is impossible for the cross-entropy algorithm to match the author to himself for the assignment.

4.3.2.4 Discussion of Experiment E7 Results

When within-assignment comparison was eliminated from Experiment E7, authorship accuracy increased by about 90 percent, from 20 correct authorship determinations to 38 correct authorship determinations, almost doubling. However, when compared with the total number of overall file comparisons, the correct classification rate is quite low, correctly assigning 38 out of 87 assignments for about 44 percent accuracy. Why the results are not quite up to 70 percent remains to be seen. Intuitively the author feels that it is because the student corpora are difficult data sets to classify. This is probably due to the small file sizes of the assignments leading to an incomplete author profile, and a lack of clear programming style yet to be developed by the students. Comparison against N-Gram later in this chapter should provide a measuring stick with which to gauge performance of cross-entropy.

With respect to Experiment E6, once again what are encouraging are the results for the last two assignments. The last two assignments, Assignments 4 and 5, comprise 33 out of the 87 files in the corpora. Out of the 38 overall correct classifications mentioned previously, 26 involve correct classifications from these two assignments, where 26 out of 33 is equal to a respectable 79 percent. Assignment 4 contained 16 source code files, 13 of which were identified correctly for an 81 percent correct classification rate. Assignment 5 contained 17 source code files, 13 of which were

identified correctly for a 76 percent correct classification rate. (What is important to remember here is that there were 18 different authors to choose from and at least 70 or more files, any of which could be selected as the best matching candidate. Randomly selecting the correct author is 1/18 or 5.5 percent.)

It is important to note that assignment similarity did play a role in the correct classifications between Assignments 4 and 5. As mentioned previously, the goal of Assignment 5 was to convert the assignment from a console application to a servlet application. All correct authorship classifications within these two assignments chose the corresponding assignment in the other assignment as the match. However, what cannot be overlooked is how impressive the algorithm was in being able to overcome the document similarity between these two assignments and still identify the correct author around 80 percent of the time, further lending credence to the idea that students are doing individual work later in the semester.

Again, it is still unclear whether the students' programming skills were beginning to evolve into an individual toolbox representing their style. However the data from Experiments E5 and E7 would seem to indicate as much, leading to a higher performance by the cross-entropy approach as the semester progresses.

What is significantly different from Experiment E6 is the correlation with file size. Figures 4.9 and 4.10 show two graphs, one depicting increases in accuracy over assignments, the other depicting file size growth over assignments as the exercises became more in-depth. As the figures show, file size was not an indicator of correct classification, further enforcing the idea that student toolbox evolution and growth are the reasons for increased classification accuracy.

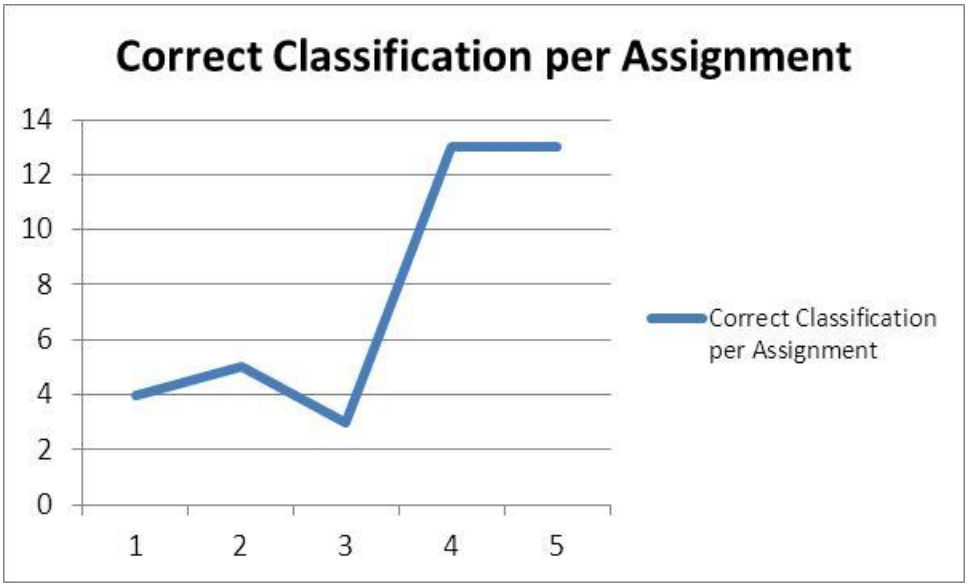

Figure 4.9 Correct Classification per Assignment for the Course 2 Corpora.

Notice Accuracy Increasing as the Assignments and Semester Progress.

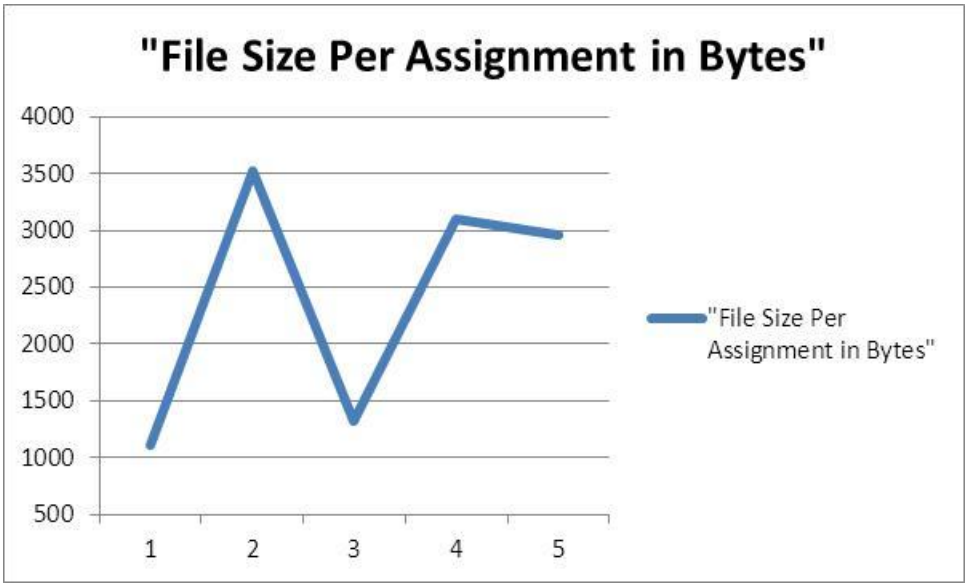

Figure 4.10 File Size per Assignment in Bytes.

Notice When Compared with Figure 4.9, File Size Shows NoCorrelation with Correct Authorship Selection.

4.3.3 Student Corpora Experiments for Course 3

The third set of experiments focuses on the corpora from Course 3, described in Section 3.1.2.3. Course 3 was composed primarily of computer science students and served as an introductory programming course using the Python programming language and C++. Note that Course 3 was a paired programming class, meaning that students were assigned a partner for most programming assignments, although the files submitted for this experiment are assumed to be from one individual's work. The Course 3 corpora contain source files from 20 student authors who submitted files for five different programming assignments, three of which are written in Python. Two of the assignments were written using C++ where the C++ assignments contained an extra credit assignment for a total of 51 files in the corpora. (Note that there will be no cross language comparisons in the experiments using the corpora with the Python files.)

For the Python experiments, using pairwise comparison for 49 files led to a total of 2352 cross-entropy experiment runs multiplied by 20 window sizes per run for a total of 47,040 comparisons. Using pairwise comparison for 51 C++ source files gave a total of 2550 cross-entropy experiment runs. Each run was computed for 20 different window sizes per run for a total of 51,000 comparisons. The two corpora combined provided a little over 98,000 comparisons.

4.3.3.1 Experiment E8 – Course 3 Experiment – Python

For Experiment E8 (Table 4.11), all Python files were compared to all other Python files within the Course 3 corpora, regardless of functionality of course assignment. The goal of this experiment was to address the same research questions explored in Experiments E4 and E6. The purpose was to provide another data set to verify results from Experiments E4 and E6, or identify patterns not previously uncovered.

Table 4.11 Summary of Experiment E8

Experiment ID	E8
Research Questions Addressed	Does the cross-entropy method provide similar accuracy results to those in Experiments E4 and E6, when applied to a student corpora whose file size samples are much smaller? Will cross-entropy accuracy increase as the semester progresses and programmers become more experienced? Will the cross-entropy approach more accurately identify authors where the test bed programs have the same functionality/objective, versus programs that have varying objectives? Will the algorithm be able to detect functional similarity?
Hypotheses	The cross-entropy approach will more accurately identify experienced programmers when compared against less experienced programmers on programs with similar objectives. Accuracy will increase as programmers gain more experience. The cross-entropy approach will less accurately identify authorship when the source files have the same functional/objective approach.
Experimental Group	Computer science students from Mississippi State University who provided source code samples for the experiment
Control Group	None
Independent Variable	None
Dependent Variable	None
Confounding Variables	Size of source code samples
Experiment Subject Population	Computer science students from Mississippi State University who provided source code samples for the experiment
Number of Subjects	20
Experiment Site	ERDC, ITL, USACE, Vicksburg, MS
Experiment Method	Subjects volunteered for the experiment and signed a consent form. Subjects emailed source code samples for five source code files determined arbitrarily by the subject of at least 150 lines of code, Source code files provided could be written by the providing author only, meaning original work, and excluded files created in tandem.
Experiment Preparations	All personal identifying information (name) of the students was stripped from the source code files and reassigned using unique identifiers for the purpose of anonymity. Source code files were divided into different directories for parallel processing in different shell scripts.
Required Resources	Anonymized source code files An implementation of the cross-entropy algorithm A processing program to generate the results through pairwise comparisons at incremental window sizes An analysis program to sift through the results identifying the most likely candidate author A MacPro with 16GB of memory and dual quad cores. To perform the experiment, a high-end machine was desired to reduce run time.
Incentives	None

4.3.3.2 Discussion of Experiment E8 Results

Unfortunately, the Python files within the Course 3 corpora provided no meaningful results. Course 3 was a paired programming class, meaning that students were encouraged to work in pairs and share code when addressing assignments. The similarity between the documents produced almost identical source files between students, skewing the results of the experiment, and finding only within-assignment topic/functionality matches 49 out of 49 times for 100 percent within-assignment identification. Most of the Python source code samples files have the same functions, variables, layouts, etc., even across assignments.

For Experiment E8, not one author was identified at any window size using cross-entropy. The documents were too similar; so similar in fact that the algorithm scored 38 out of the 49 files (almost 80 percent) as having a 98 percent similarity score or better within assignment. The not-within-assignment test produced only 5 matches out of 49 documents.

4.3.3.3 Experiment E9 – Course 3 Python Experiment not within Assignment

As mentioned in the previous section, and also confirmed in Experiments E4, E6, and now E8, assignment similarity dominated classifications because of the copying between students. This experiment (Table 4.12) aimed to increase authorship identification accuracy by eliminating the possibility of file comparisons within assignment and helping to validate the results produced in Experiments E5 and E7. Again, this makes sense because a student can submit only one work per assignment; therefore, there is no reason to compare their work to the other code samples in the same assignment because it is impossible for the cross-entropy algorithm to match the author to himself for the assignment.

Table 4.12 Summary of Experiment E9

Experiment ID	E9
Research Questions Addressed	Does the cross-entropy method accuracy increase when within-assignment comparison is eliminated? Will the cross-entropy approach more accurately identify authors where the test bed programs have the same functionality/objective, versus programs that have varying objectives?
Hypotheses	Correct authorship classification will increase from Experiment E7, once within-assignment is eliminated.
Experimental Group	Computer science students from Mississippi State University who provided source code samples for the experiment
Control Group	None
Independent Variable	None
Dependent Variable	None
Confounding Variables	Size of source code samples
Experiment Subject Population	Computer science students from Mississippi State University who provided source code samples for the experiment
Number of Subjects	18
Experiment Site	ERDC, ITL, USACE, Vicksburg, MS
Experiment Method	Subjects volunteered for the experiment and signed a consent form. Subjects emailed source code samples for five source code files determined arbitrarily by the subject of at least 150 lines of code. Source code files provided could be written only by the providing author, meaning original work, and excluded files created in tandem.
Experiment Preparations	All personal identifying information (name) of subjects was stripped from the source code files and reassigned using unique identifiers for the purpose of anonymity. Source code files were divided into different directories for parallel processing in different shell scripts. A subset of the results, comparisons of files within assignment, will be removed from the analysis.
Required Resources	Anonymized source code files An implementation of the cross-entropy algorithm A processing program to generate the results through pairwise comparisons at incremental window sizes An analysis program to sift through the results identifying the most likely candidate author To perform the experiment, a high end machine was desired to reduce A MacPro with 16GB of memory and dual quad cores. To perform the experiment, a high-end machine was desired to reduce run time.
Incentives	None

4.3.3.4 Discussion of Experiment E9 Results

There are few results to report from this experiment. Having a 98 percent similarity score or better within assignment is debilitating to the accuracy of the

algorithm. The not-within-assignment test produced only 5 matches out of 49 documents. In order to verify the problem is with the data set, the N-Gram approach will be applied later in this chapter to provide feedback and validation to the experiments performed on the Python corpora.

4.3.3.5 Experiment E10 – Course 3 Experiment, C++ Assignments

The exclusion of the Python files from further tests leaves two C++ assignments from Course 3 for comparison. While this smaller corpus does not contain as many assignments, there are still 20 students the cross-entropy algorithm can attempt to identify, from a pool of 51 files.

In Experiment E10 (Table 4.13), all C++ files are compared with all other C++ files within the Course 3 corpora, regardless of functionality/objective of course assignment. The goal of this experiment will be to address the same research questions explored in Experiments E5 and E7. The purpose is to provide another data set to verify results from Experiments E5 and E7, or identify patterns not previously uncovered.

Table 4.13 Summary of Experiment E10

Experiment ID	E10
Research Questions Addressed	Does the cross-entropy method provide accuracy results similar to those in Experiments E5 and E7, when applied to a student corpora whose file size samples are much smaller? Will cross-entropy accuracy increase as the semester progresses and programmers become more experienced? Will the cross-entropy approach more accurately identify authors where the test bed programs have the same functionality/objective, versus programs that have varying objectives? Will the algorithm be able to detect functional similarity?
Hypotheses	The cross-entropy approach will more accurately identify experienced programmers when compared against less-experienced programmers on programs with similar objectives. Accuracy will increase as programmers gain more experience. The cross-entropy approach will less accurately identify authorship when the source files have the same functional/objective approach.
Experimental Group	Computer science students from Mississippi State University who provided source code samples for the experiment
Control Group	None
Independent Variable	None
Dependent Variable	None
Confounding Variables	Size of source code samples
Experiment Subject Population	Computer science students from Mississippi State University who provided source code samples for the experiment
Number of Subjects	20
Experiment Site	ERDC, ITL, USACE, Vicksburg, MS
Experiment Method	Subjects volunteered for the experiment and signed a consent form. Subjects emailed source code samples for five source code files determined arbitrarily by the subject of at least 150 lines of code. Source code files provided could be written only by the providing author, meaning original work, and excluding files created in tandem.
Experiment Preparations	All personal identifying information (name) of the subjects was stripped from the source code files and reassigned using unique identifiers for the purpose of anonymity. Source code files were divided into different directories for parallel processing in different shell scripts.
Required Resources	Anonymized source code files An implementation of the cross-entropy algorithm A processing program to generate the results through pairwise comparisons at incremental window sizes An analysis program to sift through the results identifying the most likely candidate author A MacPro with 16GB of memory and dual quad cores. To perform the experiment, a high-end machine was desired to reduce run time.
Incentives	None

4.3.3.6 Discussion of Experiment E10 Results

This experiment followed the results pattern of earlier experiments (before the Python testing in Experiments E8 and E9). Similar to the results of Experiment E4 and E6, authorship classification was skewed because functional objective and copying between students on assignments dominated the results. The algorithm identified only 16 authors out of the 51, at some window size, for a disappointing 31 percent. The algorithm chose a candidate within the same assignment, at some window size, 48 of the 51 times for a incredibly high 94 percent selection, rather than the correct author in a different assignment.

A pattern has become clear. The cross-entropy algorithm sometimes struggles to overcome the functional objectives and copying/code sharing between students. A question examined later is "Does cross-entropy struggle as much as N-Gram with functional objectives and document similarity with respect to authorship identification?" Unfortunately, this particular corpus does not have further assignments at the end of the semester. These C++ assignments are the last submitted; therefore, it is difficult to gauge student experience and growth from such a small sample of 2 files and an extra credit assignment.

4.3.3.7 Experiment E11 – Course 3 C++ not within Assignment

Experiment E11 (Table 4.14) aims to increase authorship identification accuracy by eliminating the possibility of file comparisons within assignment and help validate the results produced in Experiment E5 and E7. The research questions addressed and the hypothesis are the same as in previous corresponding experiments. Again, it makes practical sense because a student can submit only one work per assignment; therefore there is no reason to compare their work to the other code samples in the same

assignment because it is impossible for the cross-entropy algorithm to match the author to himself for the assignment.

Table 4.14 Summary of Experiment E11

Experiment ID	E11
Research Questions Addressed	Does the cross-entropy method accuracy increase when within-assignment comparison is eliminated? Will the cross-entropy approach more accurately identify authors where the test bed programs have the same functionality/objective, versus programs that have varying objectives?
Hypothesis	Correct authorship classification will increase from Experiment E10, once within-assignment is eliminated.
Experimental Group	Computer science students from Mississippi State University who provided source code samples for the experiment
Control Group	None
Independent Variable	None
Dependent Variable	None
Confounding Variables	Size of source code samples
Experiment Subject Population	Computer science students from Mississippi State University who provided source code samples for the experiment
Number of Subjects	20
Experiment Site	ERDC, ITL, USACE, Vicksburg, MS
Experiment Method	Subjects volunteered for the experiment and signed a consent form. Subjects emailed source code samples for five source code files determined arbitrarily by the subject of at least 150 lines of code. Source code files provided could be written only by the providing author, meaning original work, and excluding files created in tandem.
Experiment Preparations	All personal identifying information (name) of subjects was stripped from the source code files and reassigned using unique identifiers for the purpose of anonymity. Source code files were divided into different directories for parallel processing in different shell scripts. **A subset of the results, comparisons of files within assignment, will be removed from the analysis.**
Required Resources	Anonymized source code files An implementation of the cross-entropy algorithm A processing program to generate the results through pairwise comparisons at incremental window sizes An analysis program to sift through the results identifying the most likely candidate author A MacPro with 16GB of memory and dual quad cores. To perform the experiment, a high-end machine was desired to reduce run time.
Incentives	None

4.3.3.8 Discussion of Experiment E11 Results

When within-assignment comparison was eliminated from Experiment E10, authorship accuracy increased by about 84 percent, from 16 correct authorship determinations to 27 correct authorship determinations. Still, when compared to the total number of overall file comparisons, the correct classification rate was quite low, correctly assigning 27 out of 51 assignments for a 53 percent accuracy rate. Of course, there is no discussion of student programming experience and growth because of the small sample size, only 2 programs and an extra credit.

4.3.3.9 Discussion of Course 3 Corpora

The Course 3 corpora proved disappointing. It is a very difficult data set to calculate because of the lack of variability within assignments. The similarity between the Python submissions and the lack of programming assignments from the C++ submissions did not provide enough granularity to fully test the performance of the cross-entropy algorithm. To further bolster the results of the performance of the cross-entropy algorithm with more meaningful analysis, additional experiments will be carried out using the Course 4 corpora.

4.3.4 Student Corpora Experiments for Course 4

The Course 4 corpora was composed primarily of computer science students, 15 who freely submitted code, where the purpose of the course is to provide an introduction to software programming using the Java programming language. Course 4 is a different section of the same class as Course 2; therefore, the programming assignments were the same, just from a different set of students. It will be interesting to see how these experiments fare when compared to Course 2 experiment results.

There are a total of 72 source files in Course 4. Once again using pairwise comparison, this leads to a total of 5112 cross-entropy experiment runs, computed for 20 different window sizes per run, giving a total of 102,240 comparisons. Combined with the 390,000 comparisons from the previous experiments, this sums to almost 500,000 comparisons. The testing for this new corpora followed the experimental plan implemented in previous assignments, focusing on within-assignment testing and removing within assignment.

4.3.4.1 Experiment E12– Course 4 Experiment within Assignment

For Experiment E12 (Table 4.15), all files were compared with all other files within the Course 4 corpora, regardless of functionality of course assignment. The goal of this experiment was to address the same research questions explored in Experiments E4 and E6. The purpose was to provide another data set to verify results from Experiments E4 and E6, or identify patterns not previously uncovered.

Table 4.15 Summary of Experiment E2

Experiment ID	E12
Research Questions Addressed	Does the cross-entropy method provide similar accuracy results to those in Experiments E4 and E6 when applied to a student corpora whose file size samples are much smaller? Will cross-entropy accuracy increase as the semester progresses and programmers become more experienced? Will the cross-entropy approach more accurately identify authors where the test bed programs have the same functionality/objective, versus programs that have varying objectives? Will the algorithm be able to detect functional similarity?
Hypotheses	The cross-entropy approach will more accurately identify experienced programmers when compared against less-experienced programmers on programs with similar objectives. Accuracy will increase as programmers gain more experience. The cross-entropy approach will less accurately identify authorship when the source files have the same functional/objective approach.
Experimental Group	Computer science students from Mississippi State University who provided source code samples for the experiment
Control Group	None
Independent Variable	None
Dependent Variable	None
Confounding Variables	Size of source code samples
Experiment Subject Population	Computer science students from Mississippi State University who provided source code samples for the experiment
Number of Subjects	15
Experiment Site	ERDC, ITL, USACE, Vicksburg, MS
Experiment Method	Subjects volunteered for the experiment and signed a consent form. Subjects emailed source code samples for five source code files determined arbitrarily by the subject of at least 150 lines of code. Source code files provided could be written only by the providing author, meaning original work, and excluding files created in tandem.
Experiment Preparations	All personal identifying information (name) of subjects was stripped from the source code files and reassigned using unique identifiers for the purpose of anonymity. Source code files were divided into different directories for parallel processing in different shell scripts.
Required Resources	Anonymized source code files An implementation of the cross-entropy algorithm A processing program to generate the results through pairwise comparisons at incremental window sizes An analysis program to sift through the results identifying the most likely candidate author A MacPro with 16GB of memory and dual quad cores. To perform the experiment, a high-end machine was desired to reduce run time.
Incentives	None

4.3.4.2 Discussion of Experiment E12 Results

As for the results in Experiments E4 and E6, functional objective of the assignments dominated the results. The algorithm identified only 21 authors out of the 72 files for a disappointing 30 percent; however, this is up 10 percent from Experiment E6, possibly because of the smaller data set. Once again, sharing and file similarity between students skewed results where 65 out of 72 times the algorithm chose a candidate within the same assignment, at some window size, for a 90 percent same assignment selection, rather than the correct author in a different assignment.

Interestingly, just as in Experiment E6, the seven files that were not matched to the same assignment at some window size were matched to the correct author in a different assignment at every window size. In addition, unlike Experiment E6, authorship was determined not only within the last two assignments of the semester (assignments 4 and 5 only) but also in assignments 2, 3, and 4. Overall, though, the majority of assignments (14), and all 7 that matched at every window size, were from assignments 4 and 5, further lending credence to the idea that distinguishable styles evolve as the student gains experience.

In fact, assignments 4 and 5 constitute 17 of the 21 correctly chosen authors. Assignments 4 and 5 contained only 27 files, meaning 17 out of 27 correct classifications for a rate of 63 percent success in the group. Of course, and as mentioned previously, a confounding factor questioning the validity of the experiment could be that assignment 5 appears to be a translation of assignment 4, where the objective is the conversion of assignment 4 from a console application to a web servlet application.

Still, as in Experiment E6, a visual inspection of the code and the cross-entropy scores support the idea that students are doing their own work, are reusing their own

code, and developing their own mechanisms for tackling the assignment. What is needed is a comparison not within assignment, such as in Experiment E7.

4.3.4.3 Experiment E13 – Course 4 Experiment not within Assignment

This experiment (Table 4.16) aims to increase authorship identification accuracy by eliminating the possibility of file comparisons within assignment and helps validate the results produced in Experiments E5 and E7. Again, this makes sense because a student can submit only one work per assignment; therefore, there is no reason to compare their work to the other code samples in the same assignment because it is impossible, as the experiments are defined, for the cross-entropy algorithm to match the author to himself for the assignment.

Table 4.16 Summary of Experiment E13

Experiment ID	E13
Research Questions Addressed	Does the cross-entropy method accuracy increase when within-assignment comparison is eliminated? Will the cross-entropy approach more accurately identify authors where the test bed programs have the same functionality/objective, versus programs that have varying objectives? Will the algorithm be able to detect functional similarity?
Hypothesis	Correct authorship classification will increase from Experiment E12, once within-assignment comparison is eliminated.
Experimental Group	Computer science students from Mississippi State University who provided source code samples for the experiment
Control Group	None
Independent Variable	None
Dependent Variable	None
Confounding Variables	Size of source code samples
Experiment Subject Population	Computer science students from Mississippi State University who provided source code samples for the experiment
Number of Subjects	15
Experiment Site	ERDC, ITL, USACE, Vicksburg, MS
Experiment Method	Subjects volunteered for the experiment and signed a consent form. Subjects emailed source code samples for five source code files determined arbitrarily by the subject of at least 150 lines of code. Source code files provided could be written only by the providing author, meaning original work, and excluding files created in tandem.
Experiment Preparations	All personal identifying information (name) of subjects was stripped from the source code files and reassigned using unique identifiers for the purpose of anonymity. Source code files were divided into different directories for parallel processing in different shell scripts. **A subset of the results, comparisons of files within assignment, will be removed from the analysis**
Required Resources	Anonymized source code files An implementation of the cross-entropy algorithm A processing program to generate the results through pairwise comparisons at incremental window sizes An analysis program to sift through the results identifying the most likely candidate author A MacPro with 16GB of memory and dual quad cores. To perform the experiment, a high-end machine was desired to reduce run time.
Incentives	None

4.3.4.4 Discussion of Experiment E13 Results

When within-assignment comparison was eliminated from Experiment E12, authorship accuracy increased from 21 correct authorship determinations to 29 correct authorship determinations, not nearly as high a jump as in previous experiments. This

could be because in Experiment E12, correct classification occurred at every assignment, not just in the last two or three assignments. However, when compared to the total number of overall file comparisons, the correct classification rate is quite low, correctly assigning 29 out of 72 assignments for about 40 percent accuracy.

Again, the results for the last two assignments, assignments 4 and 5, are encouraging . These assignments comprise 27 out of the 72 files in the corpora. Out of the 29 correct classifications mentioned earlier, 21 involve correct classifications from these two assignments, where 21 out of 27 is equal to a respectable 78 percent. Assignment 4 contains 13 source code files, 9 of which were identified correctly for a 70 percent correct classification rate. Assignment 5, the last of the semester, contains 14 source code files, 12 of which were identified correctly for a whopping 86 percent correct classification rate. (It is important to remember that there are 15 different authors to choose from and at least 70 or more files, any of which could be selected as the best matching candidate. Randomly selecting the correct author is 1/15 or 6.7 percent.)

It is important to note that assignment similarity could have played a role in the correct classifications between assignments 4 and 5. As mentioned previously, the goal of assignment 5 was to convert the assignment from a console application to a servlet application. However, all correct authorship classifications within these two assignments chose the corresponding assignment in the other assignment as the match. What is interesting to note here is that there is a good probability assignment 5 would have had a classification rate of 100 percent if student number 5 and student number 9 had submitted assignment number 4 to their instructor. Because these students did not, there was not a match for assignment 5 to find in the assignment 4 data set.

Still, what cannot be overlooked is how impressive the algorithm was in being able to overcome the document similarity and still identify the correct author around 80 percent of the time, further lending credence to the idea that students are doing individual work later in the semester, thereby increasing classification accuracy.

4.4 Cross-entropy Experiments - Student Corpora Discussion

The algorithm performed poorly at first but increased over the last assignments, which seems to support the hypothesis:

The cross-entropy approach will more accurately identify experienced programmers when compared against less experienced programmers on programs with similar objectives.

Accuracy increased over the last three assignments to 67, 81, 79, and 78 percent. It seems to be clear now that the students' programming skills were beginning to evolve into an individual toolbox representing their style. The data from Experiments E5, E7, E11 and E13 would seem to indicate as much, leading to a higher performance by the cross-entropy approach as the semester progresses.

With respect to file size and the research question:

How does the size of the author's profiles affect performance of the approach? Will a larger profile, or larger file size, say 2000 lines of code per author, cloud the results of the algorithm, help fine-tune toward a more discernable identification, or have no difference?

Disparity in file size did not appear to be an issue, unlike the professional corpora, probably because all assignments were similar in size. The author believes the cross-entropy approach does have limitations when a corpora has file size disparity, especially

considering results from Experiments E1 and E2, but these student corpora did not display those characteristics.

The true test of the cross-entropy performance from the previous experiments will be to apply a published method that claims to classify at or near 100 percent identification accuracy. The N-Gram method should be a true measuring stick showing just how accurate cross-entropy is as an approach to source code authorship attribution.

4.5 N-Gram Experiments and Comparison to Cross-entropy

As discussed in Chapter 2, there are other nonmetric-based approaches to authorship attributions. One such method is the N-Gram based approach proposed by Frantzeskou and others in [36, 45-48]. As a brief review, the N-Gram byte level approach [36, 45-48] takes source code documents of known authorship and divides them into sequences of n-grams, while assigning a normalized frequency within a document to each gram. A table is then produced showing all n-grams in a document and their associated frequencies, sorted highest to lowest. This is known as the author's profile. A test document of "unknown authorship" is then run through the same process. The N-Gram frequency profiles from the unknown authorship documents and the candidate author documents are then compared. A threshold is then set by comparing the top L N-Grams of documents, and the tables corresponding to the top L that have the most intersects are treated as more similar. In other words, the author who has the most number of common N-Grams shared between the unknown author's profile and the candidate profiles is chosen as the solution. (The experimental design and setup for N-Gram experiments in this work can be found at the end of Section 3.2) The focus of this section is to answer the following research question:

Is cross-entropy more or less accurate than other nonfeature-based approaches when determining source code authorship, specifically, the N gram approach mentioned in Chapter 2?

For the following experiments, the N-Gram approach was applied to all of the corpora presented in this work. The purpose was to provide a direct comparison of both techniques to determine if cross-entropy performs better than, or comparable to, N Gram when applied to the same source code set.

The N-Gram approach, in this author's opinion, is somewhat state of the art, and according to Frantzeskou et al. [36], is very accurate. Tables 4.17 and 4.18, taken from [36], show the accuracy results for certain N-Gram experiments performed by Frantzeskou. Note the 100 percent accuracy for the majority of corpora tested.

Table 4.17 N-Gram Accuracy Results across Various Corpora [36]

Data Sets	No of Authors	Best Classification accuracy	Profile size(L) & n-gram size(n) used for the best accuracy
MacDonellC++	6	100%	As shown in Table1
StudentJava	8	88.5%	L=2000,2500 & n=6
OSJava1	8	100%	L=1500,2000 & n=3,4,5,6,7,8
NoComJava	8	100%	L=2000 & n=5,6,7,8
OnlyComJavas	6	100%	L=1500 & n=5 and L=2000 & n=4,5,6
OSJava2	30	96.9%	L=1500 & n=7

Table 4.18 Accuracy, N-Gram Size, and Profile Size for the Macdonell C++ Experiment Performed in [36]

Profile Size L	n-gram Size					
	3	4	5	6	7	8
500	100	100	100	98	98	98
1000	100	100	100	100	100	99
1500	100	100	100	100	99	100
2000	100	100	100	100	100	100
2500	100	100	100	100	100	100
3000	100	100	100	100	100	100

As noted in Chapter 3, the experiments will be executed by adhering to the following steps:

1. Accuracy will be examined on the same data sets to provide comparison.
2. Classification within assignment (functional object) and not within assignment will be measured.
3. Pairwise comparisons will be used (this is consistent with the cross-entropy experiments).
4. Classification accuracy over semester length (experience) will also be looked at.
5. File size and accuracy issues will be examined across the corpora.
6. Ambiguity of identification (see Section 4.6) inherent to N-Gram

It is important to remember the parameter boundaries for the following experiments in this section, which tried to mimic the parameters of experiments in Frantzeskou's [36,45] work. N-Grams sizes were set at 3, 4, and 5, and profile cutoff points were set at the top 200, 500, and 1000, sorted by frequency of occurrence. The only reason the L size was not increased further is because 1000 is about the higher

threshold for most of the larger assignments, i.e. some assignments did not have more than 1000 unique N-Grams. Also, as mentioned in Chapter 3, this work proposes using only one file as the profile using pairwise comparison, not a concatenation of files to create the profile, because of the limited number of data points (files) within the corpora and to remain consistent with the cross-entropy experimental design and execution.

The remainder of this section will discuss the results from the application of the N-gram approach to all corpora previously tested by the cross-entropy approach.

4.6 Discussion of N-Gram Ambiguity and Multiple Classification

Having never implemented the N-Gram approach, a serious issue arose with regard to the definition of accuracy when determining authorship that deserves attention. Figure 4.11 shows as an example the N-Gram output for Course 1 corpora, student 15, assignment number 6 (the last of the semester), with L = 500 (L is the boundary decided upon for the top). (Figure 4.11 is generated from n-gram analysis files ranked by frequency. Figure 4.12 shows an example of two outputs for student 15). From Figure 4.12, notice that student 14 shared the highest number of N-Grams with student 15, having 496 in common, and assignment 6 was the top match for the first 8 classifications. Assignment 5 from student number 9 was the first match not within the assignment, which is incorrect. The correct classification would have been row 24 for student 15, assignment number 5.

Here is the problem. Note rows 7 and 8 in Figure 4.12, or rows 9 and 10. The intersecting N-Gram values happen to add up to the same number. Of course, in this particular example there was not more than one top candidate having the same number of common N-Grams. However, a scenario can easily be imagined where two or more top

author's intersecting N-Gram values would be the same, which would lead to ambiguity. Therefore it should be noted that a major issue with the N-Gram approach is the lack of a definitive author in some cases. (While this could also happen with the cross-entropy approach, the probability of having two or more mean match lengths calculated to the same value is quite small considering the level of precision from the division; the number of varying values for summations; and a varying divisor, which is number of times the window slides being a function of document length. For an example output, see Table 3.2.)

Assignment 5 Student 15		Assignment 6 Student 15	
FIRST N-GRAM: c s e		FIRST N-GRAM: c s e	
LAST N-GRAM: } \n \n		LAST N-GRAM: \n } \n	
_ _ _	0.0316357877463794	_ _ _	0.0329381679707587
) ; \n	0.0139098246484382	o u r	0.0145372673828749
o u r	0.0137953404949532	) ; \n	0.0130362875766125
f ("	0.0127649831135874	\n _ _	0.0116047975761956
i n t	0.0116964643477265	u r s	0.0115492057315192
\n _ _	0.0114484153485088	f ("	0.0112573485469682
u r s	0.0105134614283806	i n t	0.0103400831098078
p r i	0.00913965158655956	p r i	0.00851945019665615
r i n	0.00900608674082696	; \n \t	0.0083387767014579
n t f	0.00900608674082696	r i n	0.00813030728392145
t f (	0.00881527981835181	n t f	0.00813030728392145
; \n \t	0.00833826251216394	t f (	0.00786624602170862
v e r	0.00770859966799595	; \n _	0.00697677650688644
u i z	0.00768951897574844	a t _	0.00696287854571734
a g e	0.00740330859203572	_ = _	0.00692118466221005
e r a	0.0073842278997882	0 ; \n	0.006337470293108
r a g	0.0073842278997882	f l o	0.00621238864258613
q u i	0.00679272644011525	o a t	0.00621238864258613
_ = _	0.00665916159438264	l o a	0.00621238864258613
; \n _	0.00643019328741247	H o u	0.00617069475907884
") ;	0.00597225667347211	h o u	0.0059483273803733
0 ; \n	0.00591501459672957	v e r	0.00565647019582227
a v e	0.00589593390448205	u i z	0.00560087835114589
" , _	0.00580053044324448	e r a	0.00542020485594763
a t _	0.00566696559751188	a g e	0.00539240893360944
\\ n "	0.0055715621362743	r a g	0.00537851097244034
h o u	0.00555248144402679	" , _	0.00519783747724209

Figure 4.11 Example N-Gram Analysis Output from Two Source Files Where N-Gram Size = 3.

1	cse1233-6-14.c-nGramResults.txt	496
2	cse1233-6-16.c-nGramResults.txt	495
3	cse1233-6-13.c-nGramResults.txt	493
4	cse1233-6-12.c-nGramResults.txt	490
5	cse1233-6-10.c-nGramResults.txt	489
6	cse1233-6-17.c-nGramResults.txt	482
7	cse1233-6-3.c-nGramResults.txt	479
8	cse1233-6-4.c-nGramResults.txt	479
9	cse1233-5-9.c-nGramResults.txt	476
10	cse1233-6-1.c-nGramResults.txt	476
11	cse1233-5-7.c-nGramResults.txt	472
12	cse1233-6-5.c-nGramResults.txt	469
13	cse1233-5-6.c-nGramResults.txt	468
14	cse1233-5-5.c-nGramResults.txt	467
15	cse1233-6-6.c-nGramResults.txt	467
16	cse1233-5-3.c-nGramResults.txt	464
17	cse1233-5-4.c-nGramResults.txt	464
18	cse1233-6-7.c-nGramResults.txt	463
19	cse1233-5-2.c-nGramResults.txt	460
20	cse1233-6-8.c-nGramResults.txt	460
21	cse1233-6-9.c-nGramResults.txt	458
22	cse1233-5-17.c-nGramResults.txt	447
23	cse1233-5-16.c-nGramResults.txt	444
24	cse1233-5-15.c-nGramResults.txt	443
25	cse1233-5-14.c-nGramResults.txt	437

Figure 4.12 Ranking of Intersecting N-Grams at L = 500 for N-Gram Size = 3. Notice the Ambiguity in Rows 7 And 8. Which Is the Next Best Candidate?

A fundamental question for the N-Gram experiments becomes "Should a correct classification be considered correct if it has the highest number of intersecting N-Grams, but is one of many matches with an equivalent highest value?" Obviously, if a researcher were trying to pick "the" winner, this would not suffice. However, if a researcher were trying to pick "a" winner, it might. This could be analogous to betting on a horse race and picking half of the field as a winner to hedge the bet. However, if in a

court of law where a lawyer was trying to prove something beyond a "shadow of doubt," a scenario such as this would fail miserably.

4.7 N-Gram Experiments

The following sections will detail the experiments and results for the N-Gram approach applied to the corpora defined in Chapter 3. The first sub sections will detail experiments agains the professional corpora. The remaining sub sections will focus on student corpora experiments.

4.7.1 Experiment E14 – N-Gram Approach Applied to Professional Corpora for Comparison

As previously documented in Chapter 2, N-Gram [36, 46-48] is a state-of-the-art nonmetric-based, authorship attribution technique with high classification scores, approaching 100 percent accuracy at certain n-gram lengths. The focus of Experiment E14 (Table 4.19) is to perform N-Gram comparison against the professional corpora presented in Experiment E1 for a comparison of accuracy in order to answer the following research question:

Is cross-entropy more or less accurate than other nonfeature-based approaches when determining source code authorship, such as the N gram approach mentioned in Chapter 2?

In order to provide consistent results, the N-gram perl programming package presented by Keselj [47], and employed by Frantzeskou [36, 46, 48] (referenced in Chapter 2), was employed in this experiment and applied to the professional corpus used in Experiment E1.

Table 4.19 Summary of Experiment E14

Experiment ID	E14
Research Questions Addressed	Is the N-Gram approach more or less accurate than the cross-entropy approach when compared against the same corpora?
Hypothesis	The N-Gram approach will have a higher classification rate than the cross-entropy method.
Experimental Group	Computer programmers who provided source code samples for the experiment
Control Group	None
Independent Variable	None
Dependent Variable	None
Confounding Variables	Size of source code samples
Experiment Subject Population	Professional programmers at the U.S. Army Engineer Research Development Center, Information Technology Laboratory, Vicksburg, MS
Number of Subjects	5
Experiment Site	U.S. Army Engineer Research Development Center, Information Technology Laboratory, Vicksburg, MS
Experiment Method	Subjects volunteered for the experiment and signed a consent form. Subjects emailed source code samples for five source code files determined arbitrarily by the subject of at least 150 lines of code. Source code files provided could be written only by the providing author, meaning original work, and excluding files created in tandem.
Experiment Preparations	All personal identifying information (name) of subjects was stripped from the source code files and reassigned using unique identifiers for the purpose of anonymity. Source code files were divided into different directories for parallel processing in different shell scripts.
Required Resources	Anonymized source code files An implementation of the cross-entropy algorithm A processing program to generate the results through pairwise comparisons at incremental window sizes An analysis program to sift through the results identifying the most likely candidate author A MacPro with 16GB of memory and dual quad cores. To perform the experiment, a high-end machine was desired to reduce run time.
Incentives	None

4.7.2 Discussion of Experiment E14 Results

The N-Gram experiment E14 on the professional corpora yielded results comparable to the cross-entropy approach (Table 4.20), but certainly less than the 100 percent classification rates found in [36,45]. The N-Gram approach on average classified 18 source files correctly based on looking at all gram sizes and L profile sizes (nine different combinations where gram size can be [3,4,5] and profile size [200,500,100]), with a high of 19 when gram size = 5 and L = 1000, and a low of 17 when gram size = 3, L = 200. With respect to ambiguity, there was only one ambiguous result in each of the profile sizes of 200, 500, and 1000, with 1 at gram size = 5, 1 at gram size = 4, and 1 at gram size 3, respectively. The best match, 19 out of 25 results, is a 76 percent accuracy rate, and an average rate of 18 out of 25 for 72 percent accuracy. This is far superior to results for Experiments E1 and E2, but very close, although slightly less than the cross-entropy approach that found 18 out of 23 for 78 percent in Experiment E3. However, remember that Experiment E3 had to remove two files from the corpora because of their size in order to increase results. This appears to be a limitation of cross-entropy. N-Gram had no such problem with file size distribution. This is probably because N-Gram takes the top common N-Grams normalized, not treating the candidate as an all-inclusive reference, as does cross-entropy where anything can be found for a match. Table 4.6 shows results from the experiment.

Table 4.20 Experiment E14 N-Gram Classification Results on the Professional Corpora

Experiment E14 N-Gram Classification Results on the Professional Corpora									
25 Files	L =200			L=500			L=1000		
Gram size	n=3	n=4	n=5	n=3	n=4	n=5	n=3	n=4	n=5
Professional Corpora	17 a=0	18 a=0	18 a=1	19 a=1	17 a=0	17 a=0	18 a=0	18 a=1	19 a=0
a = ambiguous results.									

4.7.3 N-Gram applied to the Student Corpora

In this section, the N-Gram approach was applied to the student corpora source files. Issues related to functionality, experience, file sizes, and limited data were examined and compared with the results generated by cross-entropy in the previous sections of Chapter 4. At the end of the section, there will be a discussion about the results as well as a tabulation showing accuracy.

4.7.3.1 Experiment E15 – N-Gram Course 1 Experiment within Assignment

Experiment E15 (Table 4.21) compared all files to all other files within the Course 1 corpora, regardless of functionality of course assignment. The goal of this experiment was to gauge the authorship identification accuracy of the N-Gram approach when applied to the student corpora. The purpose was to provide another data set to compare cross-entropy accuracy results to N-Gram results run on the same data set.

Table 4.21 Summary of Experiment E15

Experiment ID	E15
Research Questions Addressed	Is cross-entropy more or less accurate than other nonmetric-based approaches when determining source code authorship, such as the N-Gram approach mentioned in Chapter 2? Will the N-Gram approach more accurately identify authors where the test bed programs have the same functionality/objective, versus programs that have varying objectives?
Hypothesis	N-Gram should perform better than cross-entropy when applied to the same corpora, based on references to previous works performed by others.
Experimental Group	Computer science students from Mississippi State University who provided source code samples for the experiment
Control Group	None
Independent Variable	None
Dependent Variable	None
Confounding Variables	Size of source code samples
Experiment Subject Population	Computer science students from Mississippi State University who provided source code samples for the experiment
Number of Subjects	17
Experiment Site	ERDC, ITL, USACE, Vicksburg, MS
Experiment Method	Subjects volunteered for the experiment and signed a consent form. Subjects emailed source code samples for five source code files determined arbitrarily by the subject of at least 150 lines of code. Source code files provided could be written only by the providing author, meaning original work, and excluding files created in tandem.
Experiment Preparations	All personal identifying information (name) of subjects was stripped from the source code files and reassigned using unique identifiers for the purpose of anonymity. Source code files were divided into different directories for parallel processing in different shell scripts.
Required Resources	Anonymized source code files An implementation of the cross-entropy algorithm A processing program to generate the results through pairwise comparisons at incremental window sizes An analysis program to sift through the results identifying the most likely candidate author A MacPro with 16GB of memory and dual quad cores. To perform the experiment, a high-end machine was desired to reduce run time
Incentives	None

4.7.3.2 Discussion of Experiment E15 Results

The first N-Gram experiment on Course 1 corpora yielded results comparable to those of the cross-entropy approach, but certainly less than the 100 percent classification rates found in [36,45]. The N-Gram approach on average classified 10 source files correctly, when looking at all gram sizes and L profile sizes (9 different combinations

where gram size can be [3,4,5] and profile size [200,500,100]), with a high of 13 when gram size = 4 and L = 500, and a low of 10 when gram size =3, L = 1000. The best match, 13 out of 81 results, is a 16 percent accuracy rate, with an average rate of 12 out of 81 for 15 percent. This is very close, although slightly less than the cross-entropy approach that found 14 out of 81.

Just as for the cross-entropy approach, document functionality/assignment objective or student code sharing dominated the N-Gram results. Out of 81 files, N-Gram classified the same assignment as the best match for, on average, 69 of the 81 files for a high rate of 85 percent. Also, in this experiment, only one correct classification had an ambiguous result. It was found at gram size = 3 and L = 200. (This would not be considered an influencing factor on results.)

4.7.3.3 Experiment E16 – N-Gram Course 1 Experiment outside Assignment

Experiment E16 (Table 4.22) aimed to increase authorship identification accuracy for N-Gram by eliminating the possibility of file comparisons within assignment. This approach is identical to the approach taken in the cross-entropy experiments. Again, this makes sense because a student can submit only one work per assignment. Therefore there is no reason to compare their work to the other code samples in the same assignment because it is impossible, as the experiments are set, for the N-Gram algorithm to match the author to himself for the assignment.

The goal of this experiment was to gauge the authorship identification accuracy of the N-Gram approach when applied to the student corpora. The purpose is to provide another data set to compare cross-entropy accuracy results to N-Gram results run on the same data set.

Table 4.22 Summary of Experiment E16

Experiment ID	E16
Research Questions Addressed	Is cross-entropy more or less accurate than other nonmetric-based approaches when determining source code authorship, such as the N-Gram approach mentioned in Chapter 2? Will N-Gram accuracy increase as the semester progresses and programmers become more experienced? Will the N-Gram approach more accurately identify authors where the test bed programs have the same functionality/objective, versus programs that have varying objectives?
Hypothesis	N-Gram should perform better than cross-entropy when applied to the same corpora, based on references to previous works performed by others.
Experimental Group	Computer science students from Mississippi State University who provided source code samples for the experiment
Control Group	None
Independent Variable	None
Dependent Variable	None
Confounding Variables	Size of source code samples
Experiment Subject Population	Computer science students from Mississippi State University who provided source code samples for the experiment
Number of Subjects	17
Experiment Site	ERDC, ITL, USACE, Vicksburg, MS
Experiment Method	Subjects volunteered for the experiment and signed a consent form. Subjects emailed source code samples for five source code files determined arbitrarily by the subject of at least 150 lines of code. Source code files provided could be written only by the providing author, meaning original work, and excluding files created in tandem.
Experiment Preparations	All personal identifying information (name) for subjects was stripped from the source code files and reassigned using unique identifiers for the purpose of anonymity. Source code files were divided into different directories for parallel processing in different shell scripts.
Required Resources	Anonymized source code files An implementation of the cross-entropy algorithm A processing program to generate the results through pairwise comparisons at incremental window sizes An analysis program to sift through the results identifying the most likely candidate author A MacPro with 16GB of memory and dual quad cores. To perform the experiment, a high-end machine was desired to reduce run time.
Incentives	None

4.7.3.4 Discussion of Experiment E16 Results

Removing the ability to select an author within the same assignment yielded an increase from 10 to an average of 30 correct author classifications when looking at all gram sizes and L profile sizes, with a high of 37 when gram size = 3 and L = 500 and a

low of 23 at gram size = 4, L = 200. This led to an average classification rate of 30 out of 81 files for about 37 percent. This is comparable to, although slightly better than, 28 correct classifications for 35 percent by cross-entropy. However, important to note here is the ambiguity of classification. Ambiguous results were found for 8 out of 9 gram/profile sizes with the highest being gram size = 4, L = 200, and the lowest accuracy of 23/81, finding ambiguous results 5 times. The highest accuracy result had 3 ambiguous results out of the 37. Still, the average 30 correct classifications by the N-Gram is slightly better than the cross-entropy approach.

N-Gram also bolstered the theory of increased experience yielding better results. The last two assignments of the semester contained 31 files, of which 22 were correctly classified by N-Gram, for 71 percent. This goes in line with the corresponding cross-entropy experiment (E5), which found 18 in the final two assignments.

4.7.3.5 Experiment E17 – N-Gram Course 2 Experiment within Assignment

For Experiment E17 (Table 4.23), all files were compared to all other files within the Course 2 corpora, (switching from C++ in Course 1 to Java for Course 2) regardless of functionality of course assignment. The goal of this experiment was to gauge the authorship identification accuracy of the N-Gram approach when applied to the student corpora and verify results from Experiment E15. The purpose was to provide another data set to compare cross-entropy accuracy results to N-Gram results run on the same data set.

Table 4.23 Summary of Experiment E17

Experiment ID	E17
Research Questions Addressed	Is cross-entropy more or less accurate than other nonmetric-based approaches when determining source code authorship, such as the N-gram approach mentioned in chapter 2? Will the N-Gram approach more accurately identify authors where the test bed programs have the same functionality/objective, versus programs that have varying objectives?
Hypothesis	N-Gram should perform better than cross-entropy when applied to the same corpora, based on references to previous works performed by others.
Experimental Group	Computer science students from Mississippi State University who provided source code samples for the experiment
Control Group	None
Independent Variable	None
Dependent Variable	None
Confounding Variables	Size of source code samples
Experiment Subject Population	Computer science students from Mississippi State University who provided source code samples for the experiment
Number of Subjects	18
Experiment Site	ERDC, ITL, USACE, Vicksburg, MS
Experiment Method	Subjects volunteered for the experiment and signed a consent form. Subjects emailed source code samples for five source code files determined arbitrarily by the subject of at least 150 lines of code. Source code files provided could be written only by the providing author, meaning original work, and excluding files created in tandem.
Experiment Preparations	All personal identifying information (name) for subjects was stripped from the source code files and reassigned using unique identifiers for the purpose of anonymity. Source code files were divided into different directories for parallel processing in different shell scripts.
Required Resources	Anonymized source code files An implementation of the cross-entropy algorithm A processing program to generate the results through pairwise comparisons at incremental window sizes An analysis program to sift through the results identifying the most likely candidate author A MacPro with 16GB of memory and dual quad cores. To perform the experiment, a high-end machine was desired to reduce run time.
Incentives	None

4.7.3.6 Discussion of Experiment E17 Results

This N-Gram experiment yielded an average of 8 correct author classifications out of 87 for 9 percent accuracy with a high of 11 at gram size = 5 and L = 1000, and a low of 4 at gram size = 3, L = 500. This is much less than the 20 found by cross-entropy for the same corresponding experiment (E6). There was only 1 ambiguous result in this

experiment at n = 3, L = 200. Once again, and as expected, the N-Gram had very high results when functionality/ assignment objective are considered. N-Gram classified the same assignment as the best match, instead of the correct author, for an average 79 of the 87 files for a high rate of 91 percent. What is surprising is how many more authors cross-entropy was able to identify, almost twice as much. And again, as in Experiment E15, cross-entropy was slightly better at classifying authorship when the same assignment was considered. This may be an indication that cross-entropy is better suited to identify authors where functionality and objective are the same between code samples. This could be because cross-entropy looks at the entire file, including the outlying nuances that are discarded in the N-Gram approach, where only the top L are taken. Obviously, it would seem the top L n-grams are going to be repeating keywords associated with the objective of the code. For example, a program written to average student grades will probably have a variable named "grade" or "average" across most of the files.

4.7.3.7 Experiment E18 – N-Gram Course 2 Experiment outside Assignment

This experiment (Table 4.24) is a reflection of Experiment E16, only with a different corpora, which aims to increase authorship identification accuracy for N-Gram by eliminating the possibility of file comparisons within assignment. This approach is identical to the approach taken in the cross-entropy experiments.

The goal of this experiment was to gauge the authorship identification accuracy of the N-Gram approach when applied to the student corpora and provide a comparison to Experiment E16. The purpose is to provide another data set to compare cross-entropy accuracy results to N-Gram results run on the same data set.

Table 4.24 Summary of Experiment E18

Experiment ID	E18
Research Questions Addressed	Is cross-entropy more or less accurate than other nonmetric-based approaches when determining source code authorship, such as the N-gram approach mentioned in Chapter 2? Will N-Gram accuracy increase as the semester progresses and programmers become more experienced? Will the N-Gram approach more accurately identify authors where the test bed programs have the same functionality/objective, versus programs that have varying objectives?
Hypothesis	N-Gram should perform better than cross-entropy when applied to the same corpora, based on references to previous works performed by others.
Experimental Group	Computer science students from Mississippi State University who provided source code samples for the experiment
Control Group	None
Independent Variable	None
Dependent Variable	None
Confounding Variables	Size of source code samples
Experiment Subject Population	Computer science students from Mississippi State University who provided source code samples for the experiment
Number of Subjects	18
Experiment Site	ERDC, ITL, USACE, Vicksburg, MS
Experiment Method	Subjects volunteered for the experiment and signed a consent form. Subjects emailed source code samples for five source code files determined arbitrarily by the subject of at least 150 lines of code. Source code files provided could xbe written only by the providing author, meaning original work, and excluding files created in tandem.
Experiment Preparations	All personal identifying information (name) for subjects was stripped from the source code files and reassigned using unique identifiers for the purpose of anonymity. Source code files were divided into different directories for parallel processing in different shell scripts.
Required Resources	Anonymized source code files An implementation of the cross-entropy algorithm A processing program to generate the results through pairwise comparisons at incremental window sizes An analysis program to sift through the results identifying the most likely candidate author A MacPro with 16GB of memory and dual quad cores. To perform the experiment, a high-end machine was desired to reduce run time.
Incentives	None

4.7.3.8 Discussion of Experiment E18 Results

As in Experiment E16, removing the ability to select an author within the same assignment increased N-Gram identification rates to an average of 35 correct author classifications out of 87 for 40 percent, with a high of 43 at gram size = 4, L = 1000, and

a low of 30 classifications at gram size = 3, L = 500. A result of 40 percent is comparable, but slightly lower than the 44 percent rate given by cross-entropy, which found 38 matches, but higher when the highest rate is considered. However, there were some ambiguous correct classifications, and the numbers were up in this experiment. Nine of the nine combinations of gram size and profile size found ambiguous results with gram size = 5, L = 200, finding it 11 times. Still, the numbers were comparable to cross-entropy with regard to classification, even though the ambiguity numbers are higher.

When examining correct classification over the semester, when students should be more experienced, N-Gram enforced the idea by identifying 26 out of 33 for a respectable 79 percent. This is the same number cross-entropy found over the last two assignments in Experiment E7. This strengthens the argument that classification accuracy is better when programmers are more experienced.

4.7.3.9 Course 3 Corpora Experiments

Recall that the Course 3 corpora contained assignments from two different programming languages, Python and C++. In addition, Course 3 is a paired programming class, meaning students are assigned a partner for most programming assignments, although the files submitted for this experiment were assumed to be from one individual's work. Also recall that the cross-entropy algorithm scored 38 out of the 49 files (almost 80 percent) as having a 98 percent similarity score or better within assignment.

4.7.3.10 Experiment E19 – N-Gram Course 3 Python within Assignment

For Experiment E19 (Table 4.25), all 49 files Python files were compared to all other Python files within the Course 3 corpora (cross language will not examined),

regardless of functionality of course assignment. The goal of this experiment was to gauge the authorship identification accuracy of the N-Gram approach when applied to the student corpora and verify results from Experiments E14 and E16. The purpose was to provide another data set to compare cross-entropy accuracy results to N-Gram results run on the same data set.

Table 4.25 Summary of Experiment E19

Experiment ID	E19
Research Questions Addressed	Is cross-entropy more or less accurate than other nonmetric-based approaches when determining source code authorship, such as the Ngram approach mentioned in Chapter 2? Will the N-Gram approach more accurately identify authors where the test bed programs have the same functionality/objective, versus programs that have varying objectives?
Hypothesis	N-Gram, should perform better than cross-entropy when applied to the same corpora, based on references to previous works performed by others.
Experimental Group	Computer science students from Mississippi State University who provided source code samples for the experiment
Control Group	None
Independent Variable	None
Dependent Variable	None
Confounding Variables	Size of source code samples, number of files in corpora.
Experiment Subject Population	Computer science students from Mississippi State University who provided source code samples for the experiment
Number of Subjects	20
Experiment Site	ERDC, ITL, USACE, Vicksburg, MS
Experiment Method	Subjects volunteered for the experiment and signed a consent form. Subjects emailed source code samples for five source code files determined arbitrarily by the subject of at least 150 lines of code. Source code files provided could be written only by the providing author, meaning original work, excluding files created in tandem
Experiment Preparations	All personal identifying information (name) of subjects was stripped from the source code files and reassigned using unique identifiers for the purpose of anonymity. Source code files were divided into different directories for parallel processing in different shell scripts.
Required Resources	Anonymized source code files An implementation of the cross-entropy algorithm A processing program to generate the results through pairwise comparisons at incremental window sizes An analysis program to sift through the results identifying the most likely candidate author A MacPro with 16GB of memory and dual quad cores. To perform the experiment, a high-end machine was desired to reduce run time.
Incentives	None

4.7.3.11 Discussion of Experiment E19 Results

The results for this experiment showed a change in direction for the N-Gram approach. The results were also aligned with the cross-entropy Experiment E9 results. There were 0 correct classifications by N-Gram, further emphasizing the invalidity of this data set.

With respect to choosing the same assignment when classifying, the numbers were 100 percent within assignment, choosing the same assignment by a different author 49 out of 49 times for any gram size and any profile size. The variability in this data set simply was not enough to obtain meaningful results. The only positive to come from this experiment is that the N-Gram results do validate the cross-entropy approach results from Experiment E9.

4.7.3.12 Experiment E20 – N-Gram Course 3 Experiment outside Assignment

This experiment (Table 4.26) is a reflection of Experiments E16 and E18, with the purpose of increasing authorship identification accuracy for N-Gram by eliminating the possibility of file comparisons within assignment for the Python source files. Once again, this approach is identical to the approach taken in the cross-entropy experiments.

The goal of this experiment was to gauge the authorship identification accuracy of the N-Gram approach when applied to the student corpora and provide a comparison to Experiments E15 and E17. The purpose was to provide another data set to compare cross-entropy accuracy results to N-Gram results run on the same data set.

Table 4.26 Summary of Experiment E20

Experiment ID	E20
Research Questions Addressed	Is cross-entropy more or less accurate than other nonmetric-based approaches when determining source code authorship, such as the N-gram approach mentioned in Chapter 2? Will N-Gram accuracy increase as the semester progresses and programmers become more experienced? Will the N-Gram approach more accurately identify authors where the test bed programs have the same functionality/objective, versus programs that have varying objectives?
Hypothesis	N-Gram should perform better than cross-entropy when applied to the same corpora, based on references to previous works performed by others.
Experimental Group	Computer science students from Mississippi State University who provided source code samples for the experiment
Control Group	None
Independent Variable	None
Dependent Variable	None
Confounding Variables	Size of source code samples, number of files in corpora.
Experiment Subject Population	Computer science students from Mississippi State University who provided source code samples for the experiment
Number of Subjects	20
Experiment Site	ERDC, ITL, USACE, Vicksburg, MS
Experiment Method	Subjects volunteered for the experiment and signed a consent form. Subjects emailed source code samples for five source code files determined arbitrarily by the subject of at least 150 lines of code. Source code files provided could be written only by the providing author, meaning original work, and excluding files created in tandem.
Experiment Preparations	All personal identifying information (name) of subjects was stripped from the source code files and reassigned using unique identifiers for the purpose of anonymity. Source code files were divided into different directories for parallel processing in different shell scripts.
Required Resources	Anonymized source code files An implementation of the cross-entropy algorithm A processing program to generate the results through pairwise comparisons at incremental window sizes An analysis program to sift through the results identifying the most likely candidate author A MacPro with 16GB of memory and dual quad cores. To perform the experiment, a high-end machine was desired to reduce run time.
Incentives	None

4.7.3.13 Discussion of Experiment E20 Results

For Experiment E20, removing the ability to select an author within the same assignment for the Course 3 Python corpora increased correct author classifications up to

8 on average, with a high classification of 13 at gram size = 5, L = 200, and a low where gram size = 3 and L = 200. These results are better than cross-entropy, which classified only five results. However, with respect to ambiguity, this experiment yielded the most by far, with gram size = 4, L = 200 finding ambiguity 46 times and gram size = 5, L = 200 finding ambiguity 33 times. This is exactly what is expected and can be explained.

Because of the small amount of variability within the files, when profile intersections are compared and summed, many intersections will add up to the same amount because the profiles are basically the same (e.g., different author profiles will have the same top L n-grams for their profile). So, although it did find the correct candidate authors with the most intersecting n-grams as the test document, it also found many more candidate authors ranked as highly as the correct candidate. With such high ambiguity, it could be argued that it is difficult to rank the classifications as correct because of the high amount of uncertainty. In a practical setting, results like these would not be useful.

4.7.3.14 Experiment E21 – N-Gram Course 3 C++ within Assignment

For Experiment E21 (Table 4.27), the 51 C++ files from Course 3 were examined using pairwise comparison, regardless of functionality or objective, following the same paradigm implemented in previous experiments. The goal of this experiment was to gauge the authorship identification accuracy of the N-Gram approach when applied to the student corpora and verify results from Experiments E14, E16, and E18. The purpose was to provide another data set to compare cross-entropy accuracy results to N-Gram results run on the same data set. The limiting factor with this experiment is that there are

only two to three files per author, depending on whether a student submitted the extra credit assignment.

Table 4.27 Results for Experiment E21

Experiment ID	E21
Research Questions Addressed	Is cross-entropy more or less accurate than other nonmetric-based approaches when determining source code authorship, such as the N-gram approach mentioned in Chapter 2? Will the N-Gram approach more accurately identify authors where the test bed programs have the same functionality/objective, versus programs that have varying objectives?
Hypothesis	N-Gram should perform better than cross-entropy when applied to the same corpora, based on references to previous works performed by others.
Experimental Group	Computer science students from Mississippi State University who provided source code samples for the experiment
Control Group	None
Independent Variable	None
Dependent Variable	None
Confounding Variables	Size of source code samples, number of files to compare.
Experiment Subject Population	Computer science students from Mississippi State University who provided source code samples for the experiment
Number of Subjects	20
Experiment Site	ERDC, ITL, USACE, Vicksburg, MS
Experiment Method	Subjects volunteered for the experiment and signed a consent form. Subjects emailed source code samples for five source code files determined arbitrarily by the subject of at least 150 lines of code. Source code files provided could be written onlyby the providing author, meaning original work, and excluding files created in tandem.
Experiment Preparations	All personal identifying information (name) for subjects was stripped from the source code files and reassigned using unique identifiers for the purpose of anonymity. Source code files were divided into different directories for parallel processing in different shell scripts.
Required Resources	Anonymized source code files An implementation of the cross-entropy algorithm A processing program to generate the results through pairwise comparisons at incremental window sizes An analysis program to sift through the results identifying the most likely candidate author A MacPro with 16GB of memory and dual quad cores. To perform the experiment, a high-end machine was desired to reduce run time
Incentives	None

4.7.3.15 Discussion of Experiment E21 Results

Experiment E21 provided disappointing results for the N-Gram approach, for an average classification of 5 out of 51 for about 10 percent with a high of 8 correct classifications at gram size = 5, L = 200 and of low of 3 with a gram size = 3, L = 500. This is below the 16 authors identified by cross-entropy. There was only one ambiguous classification. As expected, functionality/objective classification was very high; especially considering two of the C++ files were very closely related where one file was an extra credit assignment with the purpose of extending the previous assignment. However, it could be argued that this extension should increase author identification because of the reuse of code by the author from the first assignment to the next. Still, the numbers were 88 percent within assignment, with N-Gram choosing the same assignment by a different author, rather than the correct author in a different assignment, on 45 out of 51 files.

After examining the results of this experiment, and as mentioned in the discussion of results for Experiment E17, a trend does seem to be emerging. The cross-entropy approach is classifying at a higher rate than N-Gram when experiments are within assignment, but are very close to N-Gram results when same assignment comparison is removed.

4.7.3.16 Experiment E22 – N-Gram Course 3 C++ Experiment outside Assignment

This experiment (Table 4.28) is a reflection of Experiments E16, E18, and E20, with the purpose of increasing authorship identification accuracy for N-Gram by eliminating the possibility of file comparisons within assignment. Once again, this approach is identical to the approach taken in the cross-entropy experiments.

The goal of this experiment was to gauge the authorship identification accuracy of the N-Gram approach when applied to the student corpora and provide a comparison to Experiments E16, E18 and E20. The purpose was to provide another data set to compare cross-entropy accuracy results to N-Gram results run on the same data set.

Table 4.28 Summary of Experiment E22

Experiment ID	E22
Research Questions Addressed	Is cross-entropy more or less accurate than other nonmetric-based approaches when determining source code authorship, such as the N-gram approach mentioned in Chapter 2? Will N-Gram accuracy increase as the semester progresses and programmers become more experienced? Will the N-Gram approach more accurately identify authors where the test bed programs have the same functionality/objective, versus programs that have varying objectives?
Hypothesis	N-Gram should perform better than cross-entropy when applied to the same corpora, based on references to previous works performed by others.
Experimental Group	Computer science students from Mississippi State University who provided source code samples for the experiment
Control Group	None
Independent Variable	None
Dependent Variable	None
Confounding Variables	Size of source code samples
Experiment Subject Population	Computer science students from Mississippi State University who provided source code samples for the experiment
Number of Subjects	20
Experiment Site	ERDC, ITL, USACE, Vicksburg, MS
Experiment Method	Subjects volunteered for the experiment and signed a consent form. Subjects emailed source code samples for five source code files determined arbitrarily by the subject of at least 150 lines of code. Source code files provided could be written onlyby the providing author, meaning original work, and excluding files created in tandem.
Experiment Preparations	All personal identifying information (name) of subjects was stripped from the source code files and reassigned using unique identifiers for the purpose of anonymity. Source code files were divided into different directories for parallel processing in different shell scripts.
Required Resources	Anonymized source code files An implementation of the cross-entropy algorithm A processing program to generate the results through pairwise comparisons at incremental window sizes An analysis program to sift through the results identifying the most likely candidate author A MacPro with 16GB of memory and dual quad cores. To perform the experiment, a high-end machine was desired to reduce run time
Incentives	None

4.7.3.17 Discussion of Experiment E22 Results

Eliminating the ability to select a candidate author from within the same assignment increased identification accuracy only to an average of 23 classifications out

of 51 across all combinations of gram and profile size for about 44 percent accuracy. This is up from the classification average of 5 in the previous experiment, but a little below the cross-entropy score of 27 identifications, or 53 percent. The highest number of classifications was 25 with a gram size = 5 and L = 500, while the lowest was 19 at a gram size = 3, L = 200. There were a number of ambiguous classifications appearing, with the highest being 9 found a few times at gram size = 4, L = 1000 and gram size = 4, L = 500.

4.7.3.18 Experiment E23 – N-Gram Course 4 Java within Assignment

Experiments E23 and E24 were the final experiments for N-Gram and were tested on Course 4 corpora. The 72 Java files from Course 4 were examined using pairwise comparison, regardless of functionality or objective, following the same paradigm implemented in previous experiments. The goal of this experiment was to gauge the authorship identification accuracy of the N-Gram approach when applied to the student corpora and verify results from Experiments E15, E17, E19, and E21. The purpose was to provide another data set to compare cross-entropy accuracy results to N-Gram results run on the same data set.

4.7.3.19 Discussion of Experiment E23 Results

Experiment E23 (Table 4.29) provided disappointing results for the N-Gram approach, for an average classification of 6 out of 72 for about 9 percent accuracy with a high of 11 correct classifications at gram size = 3, L = 1000, and a low of 3 with a gram size = 3, L = 200. This is well below the 21 author files identified by cross-entropy. There were no ambiguous identifications within the experiment. With respect to same assignment matching over author, the numbers were 91 percent within assignment, with

N-Gram choosing the best match as the same assignment by a different author on average 66 times out of the of 72 file tests.

Table 4.29 Summary of Experiment E23

Experiment ID	E23
Research Questions Addressed	Is cross-entropy more or less accurate than other nonmetric-based approaches when determining source code authorship, such as the N-gram approach mentioned in Chapter 2? Will N-Gram accuracy increase as the semester progresses and programmers become more experienced? Will the N-Gram approach more accurately identify authors where the test bed programs have the same functionality/objective, versus programs that have varying objectives?
Hypothesis	N-Gram should perform better than cross-entropy when applied to the same corpora, based on references to previous works performed by others.
Experimental Group	Computer science students from Mississippi State University who provided source code samples for the experiment
Control Group	None
Independent Variable	None
Dependent Variable	None
Confounding Variables	Size of source code samples, number of files to compare.
Experiment Subject Population	Computer science students from Mississippi State University who provided source code samples for the experiment
Number of Subjects	20
Experiment Site	ERDC, ITL, USACE, Vicksburg, MS
Experiment Method	Subjects volunteered for the experiment and signed a consent form. Subjects emailed source code samples for five source code files determined arbitrarily by the subject of at least 150 lines of code. Source code files provided could be written only by the providing author, meaning original work, and excluding files created in tandem
Experiment Preparations	All personal identifying information (name) of subjects was stripped from the source code files and reassigned using unique identifiers for the purpose of anonymity. Source code files were divided into different directories for parallel processing in different shell scripts.
Required Resources	Anonymized source code files An implementation of the cross-entropy algorithm A processing program to generate the results through pairwise comparisons at incremental window sizes An analysis program to sift through the results identifying the most likely candidate author A MacPro with 16GB of memory and dual quad cores. To perform the experiment, a high-end machine was desired to reduce run time.
Incentives	None

4.7.3.20 Experiment E24 – N-Gram Course 4 Java Experiment outside Assignment

The purpose of this experiment (Table 4.30) was to increase authorship identification accuracy for N-Gram by eliminating the possibility of file comparisons within assignment. Once again, this approach is identical to the approach taken in the cross-entropy experiments.

The goal of this experiment was to gauge the authorship identification accuracy of the N-Gram approach when applied to the student corpora and provide a comparison to Experiments E16, E18, E20, and E22. The purpose was to provide another data set to compare cross-entropy accuracy results to N-Gram results run on the same data set.

Table 4.30 Summary of Experiment E24

Experiment ID	E24
Research Questions Addressed	Is cross-entropy more or less accurate than other nonmetric-based approaches when determining source code authorship, such as the N-gram approach mentioned in Chapter 2? Will N-Gram accuracy increase as the semester progresses and programmers become more experienced? Will the N-Gram approach more accurately identify authors where the test bed programs have the same functionality/objective, versus programs that have varying objectives?
Hypothesis	N-Gram should perform better than cross-entropy when applied to the same corpora, based on references to previous works performed by others.
Experimental Group	Computer science students from Mississippi State University who provided source code samples for the experiment
Control Group	None
Independent Variable	None
Dependent Variable	None
Confounding Variables	Size of source code samples
Experiment Subject Population	Computer Science students from Mississippi State University who provided source code samples for the experiment
Number of Subjects	20
Experiment Site	ERDC, ITL, USACE, Vicksburg, MS
Experiment Method	Subjects volunteered for the experiment and signed a consent form. Subjects emailed source code samples for five source code files determined arbitrarily by the subject of at least 150 lines of code. Source code files provided could be written only by the providing author, meaning original work, excluding files created in tandem.
Experiment Preparations	All personal identifying information (name) of subjects was stripped from the source code files and reassigned using unique identifiers for the purpose of anonymity. Source code files were divided into different directories for parallel processing in different shell scripts.
Required Resources	Anonymized source code files An implementation of the cross-entropy algorithm A processing program to generate the results through pairwise comparisons at incremental window sizes An analysis program to sift through the results identifying the most likely candidate author A MacPro with 16GB of memory and dual quad cores. To perform the experiment, a high-end machine was desired to reduce run time.
Incentives	None

4.7.3.21 Discussion of Experiment E24 Results

Eliminating the ability to select a candidate author from within the same assignment increased identification accuracy to an average of 33 out of 72 identifications, well above the 6 in the previous experiment, for about 45 percent accuracy rate. This is

also slightly better than the cross-entropy rate of 29 out of 72. The highest number of classifications was 35 with a gram size = 5 and L = 500, while the lowest was 29 at a gram size = 5, L = 200. There were a number of ambiguous classifications appearing, with the highest being 8 where gram size = 5, L = 200. With regard to experience increasing accuracy, 23 out of the final 27 source code files from the last 2 assignments were identified correctly for 85 percent.

4.8 Chapter 4 Discussion, Summary and Conclusion

The objectives of the experiments in this chapter were to answer the following research questions, starting with the hypothesis that cross-entropy when applied to source code will perform as well as cross-entropy applied to literary works, where accuracy came in at around 73 percent [10]. The main research question examined in this work is:

Can a cross-entropy approach be used to predict source code authorship? Will cross-entropy approaches taken in literary document classification and authorship identification fare similarly when compared to cross-entropy approaches applied to source code authorship identification?

The answer to this question is yes, which was shown in Experiment E3 and also shown when classifying the last assignments within the student corpora. Although most of the numbers from the student corpora are less than 73 percent accurate, the author believes this can be attributed to a lack of style from inexperienced programmers as well as functional similarity and code sharing between assignments. In addition, if the accuracy of N-Gram, the measuring stick for cross-entropy results, is compared, the results are very comparable. N-Gram is an established approach that boasts 100 percent accuracy in some works [36]; however, it struggled to reach 70 percent classification

when applied to the student corpora as well. Table 4.31 lists detailed comparisons of results across corpora for cross-entropy and n-gram experiments. See the comparison of results in Tables 4.32-4.36.

It should be noted the experiments uncovered some limiting factors of the cross-entropy approach that must be mitigated when constructing the corpora. In Experiments E1 and E2 cross-entropy performed poorly. From experiment E1, there are 25 source code files and from the results, two source code files were consistently chosen as the candidate in 19 of 25 runs, for a classification (or rather misclassification) rate of 76 percent. Strangely, those two files were the largest and smallest files in the professional corpora, and well outside the average file size distribution. That leads into the next set of research questions.

Table 4.31 Comparison of Results for the N-Gram and Cross-entropy Experiments, When Testing within the Same Assignment Is Eliminated

Eliminate Within Assignment	Cross-entropy Accuracy	L = 200			L = 500			L = 1000		
		n=3	n=4	n=5	n=3	n=4	n=5	n=3	n=4	n=5
Professional Corpora 25 Files	18/23 78%* 13/25 52%	17 68% a=0	18 72%a=0	18 72%a=1	19 76% a=1	17 68% a=0	17 68% a=0	18 72% a=0	18 72% a=1	19 76% a=0
Course 1 Corpora 81 C++ Files	28/81 35%	25 31% a=4	23 28% a=5	27 33% a=3	37 46% a=3	32 40% a=0	29 36% a=1	37 46% a=3	33 41% a=1	31 38% a=1
Course 2 Corpora 87 Java Files	38/87 44%	33 38% a=7	31 36% a=3	37 43% a=11	30 34% a=2	34 39% a=3	36 41% a=3	35 40% a=3	43 49% a=1	39 45% a=1
Course 3 Corpora 49 Python Files	5/49 10%	5 10% a=0	13 27% a=46	13 27% a=33	8 16% a=1	6 12% a=0	6 12% a=0	6 12% a=0	5 10% a=0	7 14% a=1
Course 3 Corpora 51 C++ Files	27/51 53%	19 37% a=4	23 45% a=3	24 47% a=1	20 39% a=4	24 47%a=9	25 49% a=7	20 39% a=4	24 47% a=9	25 49% a=7
Course 4 Corpora 72 Java Files	29/72 40%	33 46% a=3	32 44% a=4	29 40% a=8	31 43% a=3	34 47%a=6	35 49% a=2	32 44% a=0	34 47% a=1	33 46% a=1

Note: a = ambiguous classification
* 2 files were removed in E3 that were significantly beyond file size distribution.

Will a larger profile, or larger file size, say 2000 lines of code per author, cloud the results of the algorithm or help fine-tune toward a more discernable identification, or have no difference?

How does the size of the author's profiles affect performance of the approach?

-More lines of code per author should yield better predictive results.

The larger profiles/file sizes can cause problems with classification if the profiles are significantly larger or smaller than the distribution within the corpora; otherwise, no evidence related to file size and accuracy was seen. As stated earlier, empirically the reason behind the misclassifications for the larger file is because the larger file has enough code to match most of the other source code samples, meaning it is somewhat of a superset that can be thought of as a dictionary, which contains many chances for matches when window sliding. For example, imagine having a document and using a dictionary as a reference. If every word in the document was looked up and scored positively for finding a reference, it could be assumed that the person that wrote the dictionary and the document was the same author because of the common intersection of the words (at least from the perspective of the cross-entropy algorithm). However, the dictionary was the base document, and the document used as a reference, it would score very low because the dictionary has far more words not contained in the document, thereby scoring negatively for a lower correlation. In fact, what is interesting is when the largest file, F1, is the unknown document compared with the other source code files using cross-entropy, the most common classification, although incorrectly classified, is the next largest file within the corpora at 143 kilobytes.

As for the misclassification regarding the smallest document, F2, the author believes it scored high mean match length values because it contains only the key words

found in almost all of the samples, such as using statements (these are similar to import statements in C).

This smallest file, F2, is around 50 lines long, and 13 of those lines are using statements. The author believes the issue with this particular misclassification is that the algorithm never had an opportunity to lower the mean match length because it started off scoring high correlations and finished processing quickly because of file size, all before the negative scores (or no matches) could lower the mean average.

What does this mean? The supporting evidence seems to suggest that the cross-entropy method is size dependent within the corpora. Files within a corpus may need to be distributed evenly for more accurate results. When the large and small files were removed, Experiment E3 did help validate this theory and boost accuracy to levels seen in literary works analysis [10].

Literary works analysis is different from the student source code corpora analysis because most literary works have experienced writers. This is not the case for source code from students, where learning is still an ongoing activity. However, over the last few assignments classification increased dramatically, which supported the following research hypothesis:

The cross-entropy approach will more accurately identify experienced programmers when compared against less experienced programmers on programs with similar objectives.

Table 4.32 shows classification over the last two assignments for both the N-Gram and cross-entropy approaches. The accuracy rates are higher, averaging between 70-80 percent correct classification for both cross-entropy and N-Gram. (Also supporting the idea of experience leading to higher classification is the professional corpora, which

had accuracy of 78 percent in Experiment E3. The professional corpus is composed of programmers with at least 2 years experience.) This supports the idea that the cross-entropy algorithm can discern authorship, especially if an author has a style or toolbox of reusable code/techniques, usually coming with experience, that shows up throughout the authors work.

Table 4.32 Accuracy over the Final Assignments

Last Semester Assignment Matching-Eliminate within Assignment	Cross-entropy Accuracy Over Last 2 Assignments Respectively	N-Gram Accuracy over Last 2 Assignments Combined
Course 1 Corpora 81 C++ Files	8/16 50% 10/15 67%	22/31 71%
Course 2 Corpora 87 Java Files	13/16 81% 13/17 76%	26/33 79%
Course 3 Corpora 49 Python Files	-	-
Course 3 Corpora 51 C++ Files	-	-
Course 4 Corpora 72 Java Files	9/13 70% 12/14 86%	23/27 85%

Note: Accuracy is higher over the final assignments, rather than earlier assignments when students have less experience.

A comparison of Table 4.32 with Table 4.31 shows that both N-Gram and Cross-entropy classified higher in the last two assignments than the first assignments of a semester. It should be noted that an argument could be made that the file sizes were larger at the end of the semester, which could have helped the algorithm classify higher, which is true. However, from Experiment E7 and Figures 4.9 and 4.10 (shown again below), file size was not an indicator with regard to correct classification, further enforcing the idea that student toolbox evolution, experience, and growth are the reasons for increased classification accuracy.

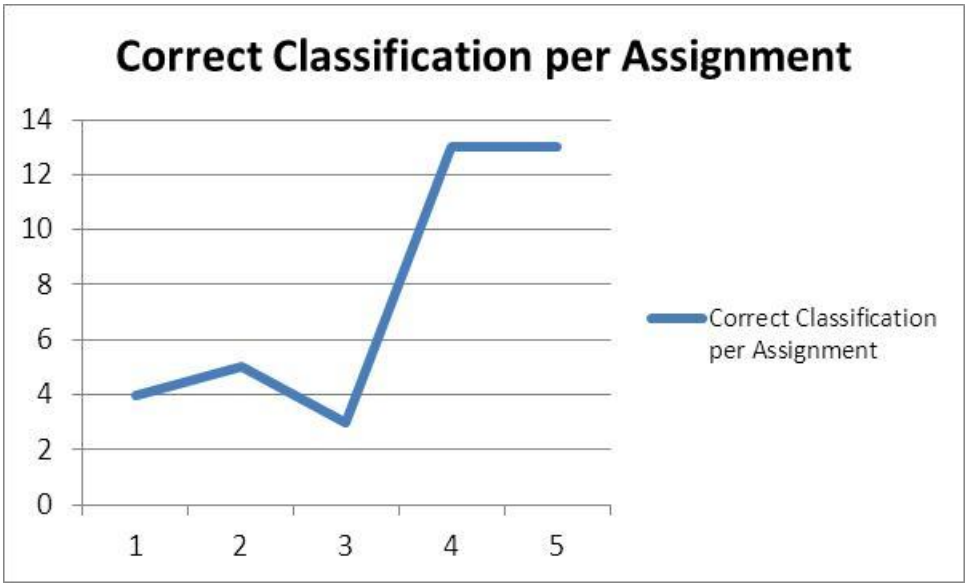

Figure 4.13 Revist of Figure 4.9. Correct classification per assignment for the Course 2 corpora.

Notice accuracy increasing as the assignments and semester progress.

The next 2 research questions can be combined into the same discussion.

Is cross-entropy more or less accurate than other nonmetric-based approaches when determining source code authorship, such as, the N gram approach mentioned in Chapter 2?

Will the cross-entropy approach more accurately identify authors where the test bed programs have the same functionality/objective, versus programs that have varying objectives?

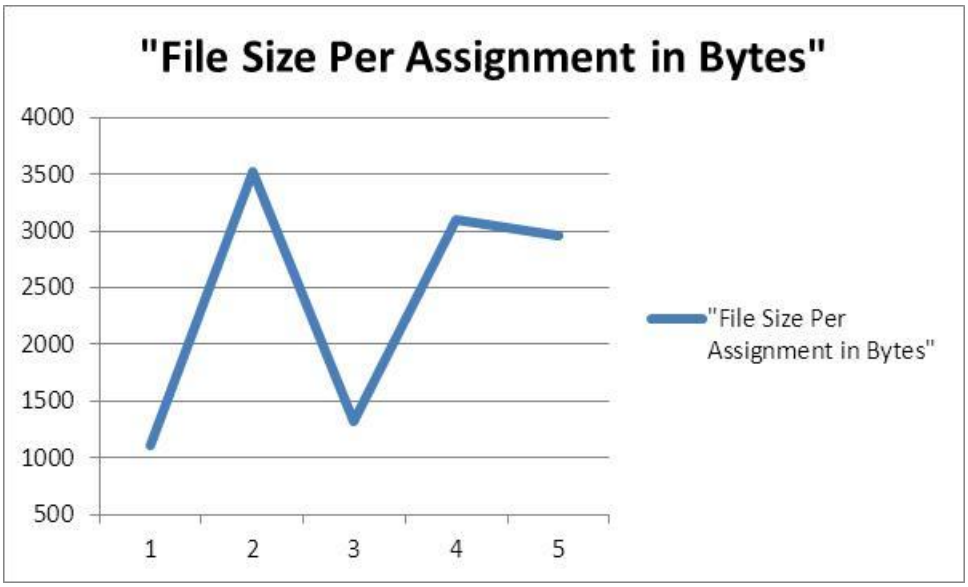

Figure 4.14 Revisit of Figure 4.10 File Size, Corpora 2.

Note: when compared with Figure 4.9, file size shows no correlation with correct authorship selection.

Cross-entropy has comparable accuracy when compared to N-Gram results. Table 4.31 shows that the accuracy rates were better for cross-entropy in the Course 3 C++ and professional corpora (E3), less accurate for Course 4 corpora and Course 3 Python (although N-Gram had numerous ambiguous classifications thereby increasing accuracy), and almost equivalent everywhere else. Considering N-Gram is the measuring stick for cross-entropy, the results are encouraging.

When focusing on the research question "*Will cross-entropy accurately identify authors where the test bed programs have the same functionality/objective, versus programs that have varying objectives"* cross-entropy classified better when within-

assignment comparison was not removed. Table 4.33 shows the misclassification rates for both cross-entropy and N-Gram when same assignment (functionality/objective) is allowed. Over 80 percent of the time the algorithms chose the same assignment/code by a different author over a different assignment/code by the same author. This is to be expected since both algorithms use distance-based calculations for solutions.

Table 4.33 Comparison of Accuracy Between Cross-Entropy and N-Gram Where Same-Assignment Testing Is Allowed

Within Assignment Testing	Cross-entropy Accuracy	L =200			L=500			L=1000		
		n=3	n=4	n=5	n=3	n=4	n=5	n=3	n=4	n=5
Course 1 Corpora 81 C++ Files	14/81 17%	11 14% a=1	12 15% a=0	12 15% a=0	10 12% a=0	13 16% a=0	13 16% a=0	10 12% a=0	12 14% a=0	13 16% a=0
Course 2 Corpora 87 Java Files	20/87 23%	7 8% a=1	6 7% a=0	7 8% a=0	4 5% a=0	7 8% a=0	6 7% a=0	10 11% a=0	10 11% a=0	11 13% a=0
Course 3 Corpora 49 Python Files	0	0	0	0	0	0	0	0	0	0
Course 3 Corpora 51 C++ Files	16/51 31%	4 8% a=0	7 14% a=1	8 16% a=0	3 6% a=0	4 8% a=0	4 8% a=0	3 6% a=0	4 8% a=0	4 8% a=0
Course 4 Corpora 72 Java Files	21/72 30%	3 4% a=0	3 4% a=0	4 6% a=0	5 7% a=0	6 8% a=0	7 10%a=0	11 15% a=0	8 11% a=0	10 14% a=0

Table 4.34 The Number Of Times Same Assignment Was Chosen over Author Identification in Experiments Where Same Assignment Comparison Is Allowed

Same Assignment Matching	Cross-entropy Accuracy	L =200			L=500			L=1000		
Course 1 Corpora 81 C++ Files	67/81 82%	71 88%	69 85%	69 85%	71 88%	68 84%	68 84%	71 88%	68 84%	67 83%
Course 2 Corpora 87 Java Files	82/87 94%	80 92%	81 93%	80 92%	83 95%	80 92%	81 93%	77 89%	77 89%	76 87%
Course 3 Corpora 49 Python Files	49/49 100%	49 100%	49 100%	49 100%	49 100%	49 100%	49 100%	49 100%	49 100%	49 100%
Course 3 Corpora 51 C++ Files	48/51 94%	45 88%	43 84%	43 84%	47 92%	46 90%	44 86%	47 92%	46 90%	44 86%
Course 4 Corpora 72 Java Files	65/72 90%	69 96%	69 96%	68 94%	67 93%	66 92%	65 90%	61 86%	64 89%	62 86%

What was surprising is cross-entropy performed substantially better on same assignment experiments than N-Gram, as shown in Table 4.33. Except for the Course 1 corpora, where the numbers were equivalent, the cross-entropy approach was twice as accurate as N-Gram. As mentioned in the discussion of Experiment E17, it is believed that this may be an indication that cross-entropy is better suited to identify authors where functionality and objective are the same between code samples. Cross-entropy looks at the entire file, including the outlying nuances. These nuances and outlying "quirks" are discarded in the N-Gram approach when the top L are taken. Obviously, when code samples are focused on the same objective, the top L n-grams are going to be repeating keywords associated with the objective of the code. For example, a program written to average student grades will probably have a variable named "grade" or "average" across most of the files. Any author characteristics are dropped in favor of grams related to variable names associated with code objective.

The final research question discussed in this section focuses on window sizes and cross-entropy accuracy.

What is the role of window size with respect to correct classification based on functional objective, experience, size, etc.?

Window size and accuracy can vary within a cross-entropy experiment run. For example, for windows sizes from 5-20, cross-entropy might recognize Author 1 as the closest match, while for window sizes 2-60 cross-entropy might recognize Author 2 as the closest match. The algorithm might then determine Author 1 is the closest match for window sizes 65-100.

However, across experiments in this work, larger window sizes tended to have higher classification rates. Table 4.35 shows the results of correct classification by window size for the cross-entropy experiments with same assignment comparison removed, i.e., eliminating functional objective misclassification. The highlighted numbers represent the highest window classification rates for experiments within the corpora.

Table 4.35 Overall Classification Accuracy per Window Size with Same-Assignment Comparison Removed.

Window Size	Course 1 Corpora	Course 2 Corpora	Course 3 C++ Corpora	Course 4 Corpora	Professional Corpora
5	21	24	12	23	16
10	24	25	11	23	17
15	23	25	11	25	15
20	21	29	13	25	13
25	21	29	14	25	14
30	20	30	14	25	14
35	21	32	16	27	14
40	21	33	17	27	14
45	21	32	21	27	14
50	22	32	20	27	14
55	22	33	21	27	15
60	21	33	20	27	15
65	21	33	20	26	15
70	21	33	21	26	15
75	21	34	21	26	15
80	21	34	21	26	16
85	21	32	21	26	16
90	21	32	21	26	16
95	21	32	21	26	16
100	20	32	21	26	16
-	-	-	-	-	-
Total 5-50	215	291	149	254	145
Total 55-100	210	328	208	262	155
-	-	-	-	-	-
Total 5-25	110	132	61	121	75
Total 30-50	105	159	88	133	70
Total 55-75	106	166	103	132	75
Total 80-100	104	162	105	130	80

Note: Window Sizes From 55-100 Usually Had The Greatest Classification Rate.

As can be seen, on average, window sizes greater than 50 had the most classifications. What is interesting to note is that in two of the corpora, the highest classification was at window size 10. It is not immediately clear why there are spikes at this window size. In general, classifications were better when window size was 55 or greater. The columns at the bottom of Table 4.35 show the summation of correct classifications per range of window sizes.

With respect to experiment results for same assignment testing, i.e., functional/objective of assignments is compared; larger window sizes were far superior. Table 4.36 shows results for the within-assignment experiments from E3 through E13, with the best classifying window sizes highlighted. The columns at the bottom of Table 4.36 show the summation of correct classifications per range of window sizes.

Table 4.36 Overall Classification Accuracy per Window Size with Same-Assignment Comparison Considered.

Window Size	Course 1 Corpora	Course 2 Corpora	Course 3 C++ Corpora	Course 4 Corpora
5	13	6	2	7
10	13	9	3	11
15	13	11	2	14
20	13	12	3	16
25	13	13	3	16
30	13	15	4	17
35	13	15	4	18
40	13	17	5	18
45	13	17	6	18
50	13	17	7	18
55	13	17	6	18
60	13	17	6	19
65	13	17	6	19
70	13	17	6	19
75	13	18	7	19
80	13	18	10	20
85	13	19	12	20
90	13	19	13	20
95	13	19	14	20
100	13	19	14	20
-	-	-	-	-
Total 5-50	130	132	39	153
Total 55-100	130	180	94	194
-	-	-	-	-
Total 5-25	65	51	13	64
Total 30-50	65	81	26	89
Total 55-75	65	86	31	94
Total 80-100	65	94	63	100

Note Window Sizes From 55-100 Usually Had The Greatest Classification Rate.

As can be seen, except for Course 1 corpora, which seems to be the anomaly, the number of correct classifications per window size increased 3-fold, 7-fold, and 3-fold, respectively, from window size 5 to 100. The author believes that higher classifications occur at larger window sizes because the commonality between the assignments (remember these results are comparing same assignments) demands a "larger" identifying marker to filter out the noise associated with smaller "shared" snippets across files. As mentioned above, when code samples are focused on the same objective, the smaller snippets are going to be repeating keywords associated with the objective of the code. For example, a program written to average student grades will probably have a variable named "grade" or "average" across most of the files. Larger window sizes find more descriptive identifiers by filtering out these repeating keywords or structures related to functionality by appending more information to the window.

Examining window size with focus on programmer experience did not reveal any new information. Assuming that the last two assignments of the semester are indicative of experience, the numbers in Table 4.37 show that larger window sizes were more accurate. Note the table is for experiments where same assignment comparison was not allowed.

Table 4.37 Overall Classification Accuracy per Window Size over the Last Two Assignments for Student Corpora Testing.

Window Size	Course 1 Corpora	Course 2 Corpora	Course 4 Corpora
5	15	17	17
10	17	19	18
15	16	19	20
20	16	20	21
25	16	19	21
30	16	20	21
35	17	22	23
40	17	23	23
45	17	23	23
50	17	23	23
55	17	23	23
60	17	23	23
65	17	23	22
70	17	23	22
75	17	24	22
80	17	24	22
85	17	24	22
90	17	24	22
95	17	24	22
100	17	24	22
-	-	-	-
Total 5-50	164	205	210
Total 55-100	170	236	222
-	-	-	-
Total 5-25	80	94	97
Total 30-50	84	111	113
Total 55-75	85	116	112
Total 80-100	85	120	110
-	-	-	-

Note That Window Sizes from 55 to100 Usually Had the Greatest Classification Rate.

CHAPTER V

CONCLUSION AND FUTURE WORK

The purpose of this research was to examine the effectiveness of the cross-entropy approach used in literary analysis [10] as a technique for source code authorship attribution in an effort to determine its viability as a valid approach. From an application viewpoint, this work is important because identification of source code authorship can be a useful tool in the area of computer security and software forensic investigation. It can help to create corroborating evidence that may send a suspected cyber terrorist, hacker, or malicious code writer to jail. When applied to academia, it can also prove as a useful tool for professors who suspect students of academic dishonesty, copying, or modification of source code related to programming assignments (as was discovered by cross-entropy when applied to all student corpora in this work). Although other methods show progress, currently no method, although some claim accuracy rates at 100 percent, is consistent across corpora (as seen with N-Gram [36] experiments in Chapter 4).

At this point, it is important to discuss issues centered around the data sets for authorship attribution experiments in general. This researcher constructed five separate corpora, with the help of others, for testing and evaluation of experiments. However, the data sets collected were not ideal. While the student corpora were interesting to examine from the standpoints of experience and programmer development, the issue with code sharing among students introduced concerns about accuracy validity since it can not be proven that an assignment is an original work by one author. A common corpus shared

by the computer forensics community is badly needed, thereby setting a baseline for results across techniques and experiments. (It should be noted that the author did in fact contact Frantzeskou [36] without success about acquiring the corpus from her experiments to give a common baseline for the experiments in this work. It should also be noted the literary works community has such a corpus to address this concern, and someone in computer forensics should step forward to volunteer. While the author would volunteer the corpora used in these experiments, he is concerned that the issues with source code copying and a lack of mature programming styles by students would serve as an insufficient data set for community use.)

Without a community shared corpora, the only logical step is to re-perform experiments on the source code samples in hand using other techniques, such as N-Gram, in order to provide a measuring stick for validation. (Of course this raises another issue related to setting up the experiments, such as whether to use pairwise comparison or n-fold cross validation, what is the size/composition of the profile, etc., which could also affect accuracy results.) The author believes future testing should consider more corpora from professional sources, where the integrity of authorship can be considered more reliable.

As for the achievements of this work, it is already known that the cross-entropy approach is successful in literary works using pairwise comparison, with a correct classification rate of about 73 percent [10]. The major goal of this research was to determine if the approach could reach classification rates this high when applied to source code, and the answer was yes, but with some caveats.

First, it must be assumed that the authors in the data set have enough programming experience to say that a unique-identifiable authorship style has evolved.

This was the assumption for Experiment E3 (professional corpora) and experiments on the last two assignments within the students' corpora (Table 4.32, with same assignment comparison removed), which showed classification rates of 78 percent, 70 percent, 86 percent, 50 percent, 67 percent, 81 percent, and 76 percent per-cent, for an overall average of 73 percent, thereby addressing the main research question of this work. This cross-sectional subset from the different corpora represents 55 authors contributing 116 different source code files, where at some window size, around 85 of the source files (compared only within corpus) were correctly identified. These results were very much in line with the N-Gram approach, the measuring stick for cross-entropy, which classified at 75 percent over 118 different source code files.

Second, an issue discovered with file size distribution, or lines of code, within a corpus skewed the accuracy results. This problem manifested when performing the first experiment on the professional corpora. One of the source code files was much larger than the rest of the distribution, acting as "dictionary" in which a snippet was always found, artificially maintaining a higher mean match length. On the other end of the spectrum was the smallest file, containing a high correlation of "using" statements with the rest of the code samples, giving it an inflated value of similarity. This issue resolved when the two files were removed. However, future testing should investigate what exactly are the boundaries for sizes, experience, etc., when creating a corpus that could negatively impact accuracy scores for cross-entropy. A hypothesis was set forth by the author that a more representative profile, i.e., an author profile containing more lines of code, would have a better chance of classifying correctly. However, this did not appear to be the case. While corpora size distribution is an issue, this author sees it as no more a

problem than n-gram sometimes ambiguously choosing two or more candidate authors with the same summation of intersecting grams.

A particular research question centered around identifying authors when the test bed programs have the same functional objective. In other words, could cross-entropy overcome the document similarity to find the nuances that describe an author's style? The results were not encouraging (Table 4.33), as the accuracy rates were quite low, around 25 percent across the student corpora where same assignment comparison was allowed, with most of the correct classifications coming later in the semester. However, these results were superior to the N-gram approach, which struggled to identify the correct author correctly 10 percent of the time, when same assignment comparison was allowed. As stated previously, this author believes that this may be an indication that cross-entropy is better suited to identify authors where functionality and objective are the same between code samples because cross-entropy looks at the entire file, including the outlying nuances. These nuances and outlying "quirks" are discarded in the N-Gram approach when the top L are taken. Obviously, when code samples are focused on the same objective, the top L n-grams are going to be repeating keywords associated with the objective of the code.

Beyond functional/objective similarity of code samples, the cross-entropy approach uses a sliding window of size n. *To further investigate accurate classification with respect to window size, the performance of the cross-entropy algorithm was explored and documented in an effort to understand the optimal window size for various code types. Across all testing and all corpora, regardless of experiences, code type, or functional objective, in general larger window sizes were more accurate (Table 4.35-4.37).*

Finally, cross-entropy was compared to an N-gram approach [36] to provide a measuring stick to gauge accuracy performance. The results were very comparable (Tables 4.31, 4.33, and 4.34) with a slight edge going to cross-entropy when same-assignment comparison is allowed. That the results were comparable is encouraging. This further validates the cross-entropy approach as a potential mechanism to support the software forensics investigative process. More testing is needed to determine the file size distribution limitations of the cross-entropy approach, as well trying different profile sizes, and experimental setups other than pairwise comparison.

REFERENCES

[1] G. Palmer, "A Road Map for Digital Forensic Research," Utica, New York 2001.

[2] A. Gray, S. MacDonell, and P. Sallis, "Software Forensics: Extending Authorship Analysis Techniques to Computer Programs," in *Proceedings of the 3rd Biannual Conference International Association of Forensic Linguistics (IAFL'97)*, 1997, pp. 1-8.

[3] P. Sallis, A. Aakjaer, and S. MacDonell, "Software Forensics: Old Methods for a New Science," in *Proceedings of SE:E&P'96*, Dunedin, New Zealand, 1996, pp. 367-371.

[4] I. Krsul and E. H. Spafford, "Authorship Analysis: Identifying the Author of a Program," Purdue University TR-96-052, 1996.

[5] T. A. Longstaff and E. E. Schultz, "Beyond Preliminary Analysis of the WANK and OILZ Worms: A Case Study of Malicious Code," *Computers and Security,* vol. 12, pp. 61-77, 1993.

[6] E. H. Spafford, "The Internet Worm Program: An Analysis," *Computer Communications Review,* vol. 19, pp. 17-49, 1989.

[7] E. H. Spafford and S. A. Weeber, "Software Forensics: Can We Track Code to its Authors," *Computers and Security,* vol. 12, pp. 585-595, 1993.

[8] P. Juola, "What Can We Do with Small Corpora? Document Categorization Via Cross-Entropy," in *Proceedings of an Interdisciplinary Workshop on Similarity and Categorization*, Edinburgh, UK, 1997.

[9] P. Juola, "Cross-Entropy and Linguistic Typology," in *Proceedings of New Methods in Language Processing 3*, Sydney, Australia, 1998.

[10] P. Juola and H. Baayen, "A Controlled-Corpus-experiment in Authorship Identification by Cross-Entropy," 2003.

[11] C. E. Shannon, "A Mathematical Theory of Communication," *Bell System Technical Journal,* vol. 27, pp. 379-423, 1948.

[12] C. E. Shannon, "Prediction and Entropy of Printed English," *Bell System Technical Journal,* vol. 30, pp. 50-64, 1951.

[13] M. Farach, M. Moordewier, S. Savari, L. Shepp, A. Wyner, and J. Ziv, "On the Entropy of DNA: Algorithms and Measurements Based on Memory and Rapid Convergence," in *Proceedings of the 6th annual Symposium on Discrete Algorithms (SODA95)*, 1995.

[14] A. J. Wyner, "Entropy Estimation and Patterns," in *Proceedings of the 1996 Workshop on Information Theory*, 1996.

[15] T. C. Mendenhall, "The Characteristic Curves of Composition," *Science,* vol. IX, pp. 237-249, 1887.

[16] G. U. Yule, *The Statistical Study of Literary Vocabulary*. Cambridge, MA: Cambridge University Press, 1944.

[17] G. U. Yule, "On Sentence-Length as a Statistical Characteristic of Style in Prose, with Application to Two Cases of Disupted Authorship," *Biometrika,* vol. 30, pp. 363-390, 1938.

[18] G. K. Zipf, *Selected Studies of the Principle of Relative Frequency in Language*. Cambridge, MA: Harvard University Press, 1932.

[19] F. Mosteller and D. L. Wallace, *Inference and Disputed Authorship: The Federalist*: Addison-Wesley, 1964.

[20] D. I. Holmes, "Authorship Attribution," *Literary and Linguistic Computing,* vol. 13, pp. 111-117, 1998.

[21] D. I. Holmes, "The Evolution of Stylometry in Humanities Scholarship," *Literary and Linguistic Computing,* vol. 13, pp. 111-117, 1998.

[22] A. Gray and S. MacDonell, "Software Forensics Applied to the Task of Discriminating Between Program Authors," *Journal of Systems Research and Information Systems* vol. 10, pp. 113-127, 2001.

[23] A. Gray, S. MacDonell, and P. Sallis, "Identified (Integrated Dictionary-Based Extraction of Non-Language-Dependent Token Information for Forensic Identification, Examination, and Discrimination): A Dictionary-Based System for Extracting Source Code Metrics for Software Forensics," in *Proceedings of SE:E&P'98 (Software Engineering: Education and Practice Conference)*, IEEE Computer Society Press, 1998, pp. 252-259.

[24] I. Krsul and E. H. Spafford, "Authorship Analysis: Identifying the Author of a Program," in *Proceedings of the 8th National Informations Systems Security Conference*, 1995, pp. 514-524.

[25] S. D. Conte, H. E. Dunsmore, and V. Y. Shen, *Software Engineering Metrics and Models*. Redwood City, CA: Benjamin-Cummings Publishing, 1986.

[26] P. W. Oman and C. R. Cook, "Programming Style Authorship Analysis," in *ACM Annual Computer Science Conference: Proceedings of the 17th Conference on ACM Annual Computer Science Conference* Louisville, Kentucky: ACM, 1989, pp. 320-326.

[27] S. Grier, "A Tool that Detects Plagiarism in Pascal Programs," in *ACM SIGCSE Bulletin*, 1981, pp. 15-20.

[28] K. J. Ottenstein, "An Algorithmic Approach to Detection and Prevention of Plagiarism," in *ACM SIGCSE Bulletin*, 1976, pp. 30-41.

[29] L. Kukolich and R. Lippmann, "LNKnet User's Guide." vol. RM E32-300. Cambridge: MIT Lincoln Library, MIT Technology Licensing Office, 1995.

[30] H. Ding and M. H. Samadzadeh, "Extraction of Java Program Fingerprints for Software Authorship Identification," *The Journal of Systems and Software Authorship Identification,* vol. 72, pp. 49-57, June 2004.

[31] R. Lange and S. Mancoridis, "Using Code Metric Histograms and Genetic Algorithms To Perform Author Identification for Software Forensics," in *Proceedings of the 9th Annual Conference On Genetic And Evolutionary Computation*, London, England, 2007.

[32] M. Shevertalov, J. Kothari, E. Stehle, and S. Mancoridis, "On the Use of Discretized Source Code Metrics for Authorship Identification," in *IEEE Proceedings of the 1st International Symposium on Search Based Software Engineering*, Windsor, UK, 2009.

[33] M. Shevertalov, E. Stehle, and S. Mancoridis, "A Genetic Algorithm for Solving the Binning Problem in Networked Applications Detection," in *IEEE Congress on Evolutionary Computation*, Singapore, 2007.

[34] J. Dougherty, R. Kohavi, and M. Sahami, "Supervised and Unsupervised Discretization of Continuous Features," in *International Conference on Machine Learning*, 1995.

[35] K. J. Kim and I. Han, "Genetic Algorithms Approach To Feature Discretization in Artificial Neural Networks for the Prediction of Stock Price Index," *Expert Systems with Applications,* vol. 19, pp. 125-132, August 2000.

[36] G. Frantzeskou, E. Stamatatos, and S. Gritzalis, "Supporting the Cybercrime Investigation Process: Effective Discrimination of Source Code Authors Based on Byte-Level Information," in *Proceedings 2nd International Conference on e-business and Telecommunications Networks (ICETE2005)*, 2005.

[37] A. Aizawa, "Linguistic Techniques To Improve the Performance of Automatic Text Categorization," in *6th Conference on National Language Processing Pacific Rim Symposium NLPRS-01*, Tokyo, Japan, 2001.

[38] S. Matwin and S. Scott, "Feature Engineering for Text Classification," in *Proceedings International Conference on Machine Learning ICML-99*, Bled, Slovenia, 1999.

[39] B. Pellin, "Using Classification Techniques to Determine Source Code Authorship," 2006.

[40] "Support Vector Machine," in *Wikipedia*, 2010.

[41] A. Moschitti, "A Study on Convolution Kernels for Shallow Semantic Parsing," in *The 42nd Annual Meeting on Association of Computational Linguistics*, Barcelona, Spain, 2004.

[42] M. Collins and N. Duffy, "New Ranking Algorithms for Parsing and Tagging: Kernels over Discrete Structures, and the Voted Perceptron," in *40th Annual Meeting on Association for Computational Linguistics*, Palo Alto, CA, 2002, pp. 263-270.

[43] "Yaxx." vol. 2010: Harvard University.

[44] G. Frantzeskou, A. MacDonell, E. Stamatatos, and S. Gritzalis, "Examining the Significance of High-Level Programming Features in Source Code Author Classification." *The Journal of Systems and Software,* pp. 447-460, 2008.

[45] V. Keselj, F. Peng, N. Cercone, and C. Thomas, "N-gram Based Author Profiles for Authorship Attribution," in *Proceedings Pacific Association for Computational Linguistics*, 2003.

[46] D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, 2nd ed., 2009.

[47] V. Keselj, "Perl package Text::Ngrams," 2003.

[48] G. Frantzeskou, E. Stamatatos, S. Gritzalis, and S. Katsikas, "Effective Identification of Source Code Authors Using Byte-Level Information," in *28th International Conference on Software Engineering*, Shanghai, China, 2006, pp. 893-896.

[49] P. Juola, "JGAAP: A System for Comparative Evaluation of Authorship Attribution," in *Chicago Colloquim of Digital Humanities and Computer Science*, Chicago, 2009.

[50] J. F. Burrows, *Computers and the Study of Literature*. Oxford: Blackwell, 1992.