Stylometric Fingerprints and Privacy Behavior in Textual Data

A Thesis
Submitted to the Faculty
of
Drexel University
by
Aylin Caliskan-Islam
in partial fulfillment of the
requirements for the degree
of
Doctor of Philosophy in Computer Science
May 2015
Dedications

To my love Seha Islam, who inspired me to follow my dream.

To the geniuses in my life, my parents Mitka and Osman Caliskan, my brother Alaattin Caliskan, and again my loving husband Seha Islam, for sharing their creativity and wisdom while always supporting me. I am so lucky to have you.

To my dear grandmother Kuna Garev, for teaching me perseverance and optimism. You are dearly missed.

To my 94 year old grandfather Rad Garev and uncle Stoyan Garev for always being there and crossing fingers for me.

To Dr. Ben Taskar, for introducing me to machine learning. Machine learning lost a star. You will always be my role model.

Finally to Dr. Rachel Greenstadt and Dr. Arvind Narayanan, two of the best things that ever happened to me. Rachel taught me the power of believing in good. Arvind inspired me to be a good scientist. I wouldn’t be here if it weren’t for you.
Acknowledgments

I would like to extend my deepest gratitude to my advisor Dr. Rachel Greenstadt for giving me the opportunity to be her Ph.D student. I have learned so many positive things from her along my journey. She helped me grow as a researcher and gain new perspectives to understand the insights behind results. She always encouraged me to pursue my curiosity, especially in source code authorship attribution. She showed me the value of believing in the greater good and always giving an opportunity to ideas, places, and people. I will forever be thankful to her for her support, optimism, invaluable ideas, and encouragement.

It is an honor to have such a great dissertation committee. I am grateful to Dr. Spiros Mancoridis, Dr. Damon McCoy, Dr. Arvind Narayanan, and Dr. Dario Salvucci for assisting me close this chapter in my life as I move on to the next one. I would also like to extend my gratitude to my candidacy exam committee members Dr. Damon McCoy, Santiago Ontanón, Dr. Dario Salvucci, and Dr. Ali Shokoufandeh, who helped me navigate more efficiently through research with valuable advice.

I would like to thank the U.S. Army Research Laboratory, especially William Glodek, Dr. Richard Harang, Dr. Melissa Holland, Jeffrey Micher, and Dr. Clare Voss for making it possible for me to work at the Adelphi Laboratory Center under the open campus program with the network security and natural language processing divisions. I would like to thank them for their hospitality and kindness. I learned many new useful things from the experts in different fields while enjoying a very productive summer. I would like to express my special appreciation to Dr. Richard Harang, whose expertise and resourcefulness in mathematics and machine learning is fascinating to me. His precision and clarity in explaining mathematical theories and machine learning is astounding. It is such an honor that we will keep working with him and the network security division in the upcoming years.
My co-author Dr. Arvind Narayanan, with his genius ideas and perfect papers, has been very motivating and inspirational during my Ph.D. His enthusiasm for science is contagious. I cannot find words to express how thankful I am for the countless hours he spent with me to come up with new approaches and to discuss new research directions and results. I could not have wished for a better mentor. I am also so lucky to have Dr. Damon McCoy as my co-author. His expertise on cybercrime and resources on underground forums have made my experiments so much fun that they have started becoming a hobby. Collaboration has become essential and very productive after seeing his openness to new directions and dedication. My co-author Fabian Yamaguchi has been very helpful with his valuable contributions and support during our collaboration on de-anonymizing programmers. Being surrounded with so many inspiring people makes research a lot more exciting and rewarding.

My senior colleagues Dr. Michael Brennan and Dr. Sadia Afroz have been very helpful since the day I met them. Mike’s presentation and writing skills have been helpful examples throughout my academic career, while submitting a paper, presenting at the Chaos Communication Congress, or at a conference. My co-author Sadia has set a great example for realizing the opportunities in problems faced during research to come up with novel approaches. My co-author, a good programmer Ariel Stolerman has always come up with effective solutions whenever we were stuck in research. Understanding privacy behavior in social networks would not have been possible without the help of my disciplined co-author Jonathan Walsh. I am lucky to have so many great labmates at the Privacy, Security, and Automation Laboratory at Drexel University, especially Rebekah Overdorf, Edwin Dauber, and Andrew McDonald. I am grateful that they will continue to carry the exciting work forward.

Finally, I would like to thank DARPA, NSF, ARO/ARL, and Amazon for providing generous funding.
Table of Contents

LIST OF TABLES ........................................................................... viii
LIST OF FIGURES ........................................................................ x
ABSTRACT ................................................................................... xii
1. Introduction ........................................................................ 1
   1.1 Statement of Thesis ....................................................... 6
   1.2 Key Contributions ......................................................... 7
   1.3 Thesis Organization ....................................................... 8
2. Linguistic Style Feature Extraction ........................................... 10
   2.1 Author Identification and Multiple Identity Detection in Cyber Criminal Forums ........................................ 11
      2.1.1 Related Work ....................................................... 15
      Underground Markets ................................................. 15
      Authorship Attribution ............................................... 16
      Detecting Fraudulent Accounts .................................... 17
      2.1.2 Overview of Underground Forums ......................... 18
      Properties of Antichat .................................................. 19
      Properties of BlackhatWorld ....................................... 21
      Properties of Carders .................................................. 22
      Properties of L33tCrew ............................................... 23
      Member Overlap Between Forums ............................... 24
      Identity Protection ..................................................... 25
      Public and Private Messages ...................................... 27
      2.1.3 Authorship Attribution ......................................... 27
      Approach ............................................................... 27
      Feature Extraction .................................................... 28
      Classification .......................................................... 29
      Removing Product Data .......................................... 29
      2.1.4 Results ............................................................. 31
      Minimum Text Requirement for Authorship Attribution .... 31
      Attribution Within Forums ....................................... 32
      Importance of Features ............................................. 34
   2.2 Detecting Multiple Identities ............................................. 35
      2.2.1 Approach ........................................................... 35
      2.2.2 Feature Extraction ............................................... 36
      2.2.3 Probability Score Calculation .................................. 37
      2.2.4 Multiple Identities in Underground Forums .......... 38
      2.2.5 Multiple Identities Across Forums ....................... 39
      2.2.6 Multiple Identities Within a Forum ...................... 40
      Methodology .......................................................... 41
      Results and Discussion ............................................. 42
      2.2.7 Automating Forum Analysis ................................... 46
2.2.8 Lessons Learned about Underground Markets ..................................................... 48
   Reasons for Creating Multiple Identities .......................................................... 49
2.2.9 Lessons Learned about Stylometry ............................................................... 50
2.2.10 Doppelgänger Detection by Forum Administrators ....................................... 50
2.2.11 Performance .................................................................................................. 51
2.2.12 Hybrid Doppelgänger Finder Methods ......................................................... 51
2.2.13 Methods to Evade Doppelgänger Finder ...................................................... 52
2.2.14 Conclusion .................................................................................................... 52
2.3 Author Identification in Translated Text .......................................................... 53
   2.3.1 Introduction .................................................................................................. 53
   2.3.2 Related Work ............................................................................................... 54
   2.3.3 Corpus Selection .......................................................................................... 55
   2.3.4 Experiment Design ...................................................................................... 57
       Translator Attribution Experiments .................................................................. 57
       Authorship Attribution Experiments .................................................................. 59
   2.3.5 Results and Evaluation ................................................................................ 59
       Translator Attribution Results ........................................................................ 59
       Translator Attribution Results in One-way Translations ............................... 60
       Authorship Attribution Results ....................................................................... 61
       Authorship Attribution Results in One-way Translations ............................... 64
   2.3.6 Conclusion ................................................................................................... 66
2.4 Author Identification in Source Code .............................................................. 67
   2.4.1 Introduction .................................................................................................. 67
   2.4.2 Problem Statement ....................................................................................... 69
   2.4.3 De-anonymizing Programmers ..................................................................... 72
       Fuzzy Abstract Syntax Trees ........................................................................... 72
       Feature Extraction ............................................................................................ 74
       Lexical and Layout Features ............................................................................. 74
       Syntactic Features ............................................................................................. 76
   2.4.4 Classification ................................................................................................ 78
       Feature Selection ............................................................................................... 79
       Random Forest Classification ......................................................................... 81
   2.4.5 Evaluation .................................................................................................... 82
       Corpus .............................................................................................................. 83
       Programmer De-anonymization ....................................................................... 85
       Scaling ............................................................................................................. 87
       Training Data and Features ............................................................................. 87
       Obfuscation ....................................................................................................... 90
       Two-class Classification .................................................................................... 92
       Verification/Open World Problem .................................................................... 94
       Relaxed Classification ....................................................................................... 96
       Generalizing the Method ................................................................................. 96
       Software Engineering Insights .......................................................................... 98
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.4.6</td>
<td>Discussion</td>
<td>100</td>
</tr>
<tr>
<td>2.4.7</td>
<td>Related Work</td>
<td>102</td>
</tr>
<tr>
<td>2.4.8</td>
<td>Conclusion and Future Work</td>
<td>104</td>
</tr>
<tr>
<td>3.1</td>
<td>Introduction</td>
<td>108</td>
</tr>
<tr>
<td>3.2</td>
<td>Related Work</td>
<td>112</td>
</tr>
<tr>
<td>3.3</td>
<td>Problem Statement and Threat Model</td>
<td>115</td>
</tr>
<tr>
<td>3.4</td>
<td>Data Collection</td>
<td>115</td>
</tr>
<tr>
<td>3.5</td>
<td>Amazon Mechanical Turk Annotations</td>
<td>118</td>
</tr>
<tr>
<td>3.6</td>
<td>Approach</td>
<td>121</td>
</tr>
<tr>
<td>3.6.1</td>
<td>Text Preprocessing</td>
<td>122</td>
</tr>
<tr>
<td>3.6.2</td>
<td>Feature Extraction</td>
<td>123</td>
</tr>
<tr>
<td></td>
<td>Feature Normalization</td>
<td>124</td>
</tr>
<tr>
<td></td>
<td>Topic Ratios</td>
<td>124</td>
</tr>
<tr>
<td></td>
<td>Privacy Dictionary Matches</td>
<td>126</td>
</tr>
<tr>
<td></td>
<td>Named Entity Recognition</td>
<td>127</td>
</tr>
<tr>
<td></td>
<td>Sentiment Analysis</td>
<td>128</td>
</tr>
<tr>
<td></td>
<td>Quote, URL, Handle, Retweet, Hashtag Count</td>
<td>128</td>
</tr>
<tr>
<td>3.7</td>
<td>Results</td>
<td>129</td>
</tr>
<tr>
<td>3.7.1</td>
<td>Twitter Database User Scores</td>
<td>130</td>
</tr>
<tr>
<td>3.7.2</td>
<td>Correlation between User's Privacy Score and User's Friends' Privacy Score</td>
<td>130</td>
</tr>
<tr>
<td>3.7.3</td>
<td>Correlation between User's Privacy Score and Mentioned Users' Privacy Score</td>
<td>133</td>
</tr>
<tr>
<td>3.7.4</td>
<td>Correlation between User's Privacy Score and Number of Followers</td>
<td>133</td>
</tr>
<tr>
<td>3.7.5</td>
<td>Inter-Annotator Agreement</td>
<td>134</td>
</tr>
<tr>
<td>3.8</td>
<td>Limitations</td>
<td>135</td>
</tr>
<tr>
<td>3.9</td>
<td>Discussion</td>
<td>136</td>
</tr>
<tr>
<td>3.9.1</td>
<td>Future Work</td>
<td>138</td>
</tr>
<tr>
<td>3.10</td>
<td>Conclusion</td>
<td>138</td>
</tr>
<tr>
<td>4.1</td>
<td>Future Directions</td>
<td>141</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Design “Nudges” and NLP-based Privacy Protection Policies</td>
<td>141</td>
</tr>
<tr>
<td>4.1.2</td>
<td>De-classification and Sanitization</td>
<td>141</td>
</tr>
<tr>
<td>4.1.3</td>
<td>Source Code and Binaries</td>
<td>142</td>
</tr>
<tr>
<td>4.2</td>
<td>Conclusion</td>
<td>143</td>
</tr>
<tr>
<td>BIBLIOGRAPHY</td>
<td></td>
<td>145</td>
</tr>
<tr>
<td>APPENDIX A: COPYRIGHT INFORMATION</td>
<td></td>
<td>155</td>
</tr>
</tbody>
</table>
List of Tables

2.1 Summary of Forums ............................................................. 19
2.2 AntiChat Members Rank ....................................................... 20
2.3 Blackhat Members Rank ......................................................... 22
2.4 Carders Members Rank ......................................................... 23
2.5 L33tCrew Members Rank ....................................................... 24
2.6 Feature Set ....................................................................... 29
2.7 Author Attribution Within a Forum. .......................................... 33
2.8 Features with Highest Information Gain Ratio in Different Forums ........ 34
2.9 Criteria for Verifying Multiple Accounts ..................................... 41
2.10 Manual Analysis of Users: X indicates same, – indicates different, empty means the result is inconclusive or complicated with many values........... 43
2.11 A Product and Quantity Cluster from Public Carders Messages ........... 47
2.12 Words Closest to the Word: 'weed' ............................................. 48
2.13 Translation Feature Set ......................................................... 59
2.14 'Translation Feature Set’ Translator Attribution .............................. 60
2.15 'Translation Feature Set’ Translator Attribution on One-way Translations . 60
2.16 'Translation Feature Set’ Authorship Attribution ............................. 61
2.17 'Translation Feature Set’ Authorship Attribution on One-way Translations 65
2.18 Overview of Applications for Code Stylometry .............................. 72
2.19 Abstract Syntax Tree Node Types ............................................. 75
2.20 Lexical Features .................................................................. 76
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.21</td>
<td>Layout Features</td>
<td>77</td>
</tr>
<tr>
<td>2.22</td>
<td>Syntactic Features</td>
<td>78</td>
</tr>
<tr>
<td>2.23</td>
<td>C++ Keywords</td>
<td>79</td>
</tr>
<tr>
<td>2.24</td>
<td>Validation Experiments</td>
<td>86</td>
</tr>
<tr>
<td>2.25</td>
<td>Effect of Obfuscation on De-anonymization</td>
<td>93</td>
</tr>
<tr>
<td>2.26</td>
<td>Generalizing to Other Programming Languages</td>
<td>98</td>
</tr>
<tr>
<td>2.27</td>
<td>Effect of Problem Difficulty on Coding Style</td>
<td>99</td>
</tr>
<tr>
<td>2.28</td>
<td>Comparison to Previous Results</td>
<td>105</td>
</tr>
<tr>
<td>3.1</td>
<td>Data Set Information</td>
<td>117</td>
</tr>
<tr>
<td>3.2</td>
<td>Tweet Privacy Categories</td>
<td>120</td>
</tr>
<tr>
<td>3.3</td>
<td>Privacy Feature Set</td>
<td>125</td>
</tr>
<tr>
<td>3.4</td>
<td>Some Private and Public Topics</td>
<td>127</td>
</tr>
<tr>
<td>3.5</td>
<td>Information Gain</td>
<td>137</td>
</tr>
<tr>
<td>3.6</td>
<td>Topics with High Information Gain</td>
<td>140</td>
</tr>
</tbody>
</table>
List of Figures

1.1 Overview of My Research....................................................... 2

2.1 Duplicate Account Groups Within Carders as Identified by the AE Detector. (Each dot is one user. There is an edge between two users if the AE detector considered them as duplicate users.) ........................................ 26

2.2 Effect of Number of Words Per User on Accuracy ..................... 31

2.3 User Attribution on 50 Randomly Chosen Authors. ....................... 33

2.4 Doppelgänger Finder: With Common Users in Carders and L33tCrew: 179 Users with 28 Pairs. AUC is 0.82. ........................................ 39

2.5 Combined Probability Scores of the Top 100 Pairs from Carders. .... 40

2.6 Comparison of the Effectiveness of ‘Translation Feature Set’ in Translator Attribution vs. Authorship Attribution ................................. 63

2.7 Comparison of Authorship Attribution Using the ‘Translation Feature Set’ and Excluding Function Words............................................ 64

2.8 Sample Code Listing ............................................................ 73

2.9 Corresponding Abstract Syntax Tree ........................................... 73

2.10 Large Scale De-anonymization................................................. 88

2.11 Training Data.................................................................... 89

2.12 A Code Sample X ................................................................ 91

2.13 Code Sample X After Obfuscation ........................................... 92

2.14 Relaxed Classification with 250 Programmers .............................. 96

3.1 AMT Annotation Results....................................................... 119

3.2 Workflow........................................................................ 122

3.3 Tweet Preprocessing............................................................ 124
Abstract
Stylometric Fingerprints and Privacy Behavior in Textual Data

Aylin Caliskan-Islam
Advisor: Rachel Greenstadt, Ph.D.

Machine learning and natural language processing can be used to characterize and quantify aspects of human behavior expressed in language. Linguistic features exhibited in any kind of text can be used to study individuals' behavior as well as to identify an author among thousands of authors. Studying aspects of human behavior can be automated by incorporating machine learning techniques and well-engineered features that represent behavior of interest. Human behavior analysis can be used to enhance security by detecting malware programmers, malicious users, or abusive multiple account holders in online networks. At the same time, such an automated analysis is a serious threat to privacy, especially to the privacy of persons that would like to remain anonymous. Nevertheless, privacy enhancing technologies can be built by first and foremost understanding privacy infringing methods in-depth to create countermeasures.

Authorship attribution through stylometry, the study of writing style, in translated or unconventional text yields as high accuracy as the state-of-the-art accuracy in authorship attribution in English prose. Applying stylometry to the more structured domain of programming languages is also possible through a robust and principled method introduced in this thesis. Code stylometry is able to de-anonymize thousands of programmers with high accuracy while providing insight into software engineering. Programmer de-anonymization can aid in forensic analysis, resolving plagiarism cases, or copyright investigations. On the other hand, de-anonymizing programmers constitutes a privacy threat for anonymous contributors of open source repositories. Bridg-
ing the gap between natural language processing and machine learning is a powerful step towards designing feature sets that represent aspects of human behavior. Features obtained through natural language processing methods can be used to study the privacy behavior of users in large social networks. Aggregate privacy analysis shows that people with similar privacy behavior appear in clusters. This knowledge can be used to design privacy nudges and effective privacy preserving technologies. Machine learning can be incorporated on any kind of textual data to automate human behavior extraction in large scale.
1. Introduction

Machine learning and natural language processing can be used to characterize and quantify aspects of human behavior expressed in language. This work touches two main realms, security and privacy. Its applications complement each other by enhancing security and preserving privacy. My work is grounded in computer science, draws on computational social science, human computer interaction, and behavioral economics, and has applications to public policy. Figure 1.1 shows a bottom-up overview of my work. My work builds upon the key element of feature extraction by incorporating language parsing, abstract syntax trees, topic modeling, word clustering, named entity recognition, and semantic classification. These features along with rigorous analysis of large textual data sets makes it possible to extract aspects of human behavior. These techniques are tailored to investigate the semi-structured domain of natural languages. I have also ported these techniques to the more structured domain of programming languages by analyzing source code.

De-anonymization This thesis introduces code stylometry, a principled and robust method for de-anonymizing programmers, in addition to advancing the state-of-the-art in stylometry and social behavior analysis by bridging the gap between natural language processing and machine learning. Experts in linguistics, forensics, and economics have been interested in de-anonymizing the decentralized digital currency Bitcoin’s anonymous founder Satoshi Nakamoto. Satoshi Nakamoto is the pseudonym of Bitcoin’s founder(s) who prefer(s) to remain anonymous. Bitcoin’s white paper has been published in 2008 and its open source code has been released in 2009. Bitcoin has attracted the attention of researchers, financial regulators, legislators, law enforcement, and criminals while Satoshi Nakamoto remained anonymous. As the
digital currency had a significant impact on the market and raised security questions, forensic experts became more interested in de-anonymizing Satoshi Nakamoto. Experts have started looking for possible candidates that can be Bitcoin’s inventor by first identifying researchers working at the intersection of computer science and digital currencies, that included computer scientists, cryptographers, and mathematicians. After identifying a set of candidates by research topics and online communications, experts have analyzed the writing styles of the candidates by using stylometry. Stylometry, the study of writing style, can unveil the candidate that has the closest writing style exhibited in Bitcoin’s white paper that was published in 2008. Some linguistic analysts have suggested that Satoshi Nakamoto is actually Nick Szabo, a legal scholar and cryptographer. Numerous people have been suggested to be Satoshi Nakamoto after performing stylometric analysis on essays, blogs, and different forms of available prose with an effort to de-anonymize this famous anonymous inventor. No one has gone one step further to analyze coding style as Satoshi’s fingerprint in Bitcoin’s initial git repository released in 2009. If we have a set of programmers who
we think might be Satoshi, and samples of source code from each of these programmers, we could use the initial versions of Bitcoin’s source code to try to determine Satoshi’s identity. Of course, this assumes that Satoshi didn’t make any attempt to obfuscate his or her coding style.

**Fingerprints**  
Internet users leave fingerprints on the Internet as they share any form of textual data. When users share textual data on social networks, they willingly reveal private information. Even in cases when personal information is not shared, users can be de-anonymized by low properties of their text through stylometric analysis. Stylometry is the study of writing style. Writing style is unique to each individual which makes it possible to identify individuals in large data sets. Advanced natural language processing and machine learning methods make it possible to profile individuals and track them even when they are careful about not revealing any private information in social media. This work focuses on two cases that carry personal fingerprints, namely textual data in social networks that contain private information and text that reveals identity through personal style.

**Stylometric Fingerprints**  
Privacy savvy people might refrain from sharing any high level personal information on social media to minimize their Internet presence. Nevertheless, their identity is still preserved in the low level properties of their text through their writing style unless they effectively obfuscate their writing. Stylometric methods for authorship attribution can successfully identify authors of anonymous documents in large data sets [99] [6] [80]. Here, we consider the supervised authorship attribution problem which relies on correct ground truth authorship information, that given a document \( D \) and a set of unique authors \( A = \{A_1, \ldots, A_n\} \), where \( A_i \neq A_j \) when \( i \neq j \), determines who among the authors in \( A \) wrote \( D \). The algorithm has two steps: training and testing. During training, the algorithm trains a classifier
from the sample documents of the authors in $A$. In the testing step, it determines the probability of each author in $A$ being the author of $D$ and assigns the author with the highest probability as the author of $D$. The success of this method depends on how well the features express the writing style of the authors in the data set. In the presence of correct features, this method can be used to identify the authors or translators in text that has been translated to other languages and back to English, which rules out the possibility of obfuscating text by translating it. Linguistic features can be used not only to infer authorship but also to detect an author’s native language [108] or find accounts owned by this author across different domains [84]. Being able to identify authors across domains facilitates linking identities across the Internet, making this a key privacy concern. On the other hand, being able to find multiple accounts of a user within a domain can help detect abusive accounts. Such stylometric techniques can even be used in challenging data sets where text is a mixture of different languages, slang, product information, and l33t-speak [10].

**Code Stylometry** Source code is a form of structured textual data that preserves personal coding style to a great extent. Source code is becoming more easily accessible as open source software, online version control and bug tracking repositories are becoming widely used. Source code authorship attribution can be constructed as an identical machine learning problem to supervised authorship attribution of anonymous documents. The general case is again a supervised authorship attribution problem that relies on correct ground truth authorship information, that given a source code file $C$ and a set of programmers $P = \{P_1, ..., P_n\}$, where $P_i \neq P_j$ when $i \neq j$, determines who among the authors in $P$ wrote $C$. A classifier is trained on features extracted from source files with known programmers. Extracting syntactic features from source code requires parsing the code to generate abstract syntax trees. Features are extracted for each programmer to generate a numeric representation of
their coding style. In the testing step, the probability of each programmer in $P$ being the programmer of $C$ is calculated and the programmer with the highest probability is assigned as the author of $C$. Code stylometry holds important implications for protecting intellectual property as well as for identifying malware authors, resolving copyright disputes or aid in plagiarism investigations. Source code authorship attribution spurs a cross-cutting area involving natural language processing and machine learning. Identifying features of coding style also reveals information about how coding style changes under certain circumstances. Modeling the coding styles of programmers, who introduce bugs or security vulnerabilities to code repositories, can be used to automatically detect problematic code.

**Privacy Behavior** Social network participants consciously share private information in public or private settings, such as location or family information. Sharing information through private channels still comes with the risk of that information being exposed to the public through the people it was shared with. As a result, enormous amounts of personal data about individuals is accumulating on the Internet and is being collected by data aggregators, which is a serious threat to privacy. Companies sell personal information for marketing purposes and sometimes for reasons that we are not even informed about. Personal information on social media can be regulated in a more informed manner by end users if they have the tools to analyze what type of an Internet identity they are forming on the Internet. Understanding the private information revealing behavior of friends in social networks can help a user decide when to share what type of information with whom. Quantifying private information can provide insight into the social dynamics of private information sharing. As private information is quantified, it can be used to associate certain privacy behaviors with users in social networks, and one example is associating a privacy score with each user to symbolize the amount of private information she shares. A machine
learning classifier can be trained on a set of textual data $\mathcal{T} = \{T_1, ..., T_n\}$ with known privacy score, to predict the privacy scores of users $\mathcal{A} = \{A_1, ..., A_n\}$ by extracting features that reveal private information. Quantifying privacy behavior can show how it is constructed and influenced, and this knowledge can be used to effectively design privacy enhancing technologies and target educational interventions. Awareness on privacy behavior can help users avoid posts that they later regret which might cause loss of jobs or relationships.

Advanced natural language processing and machine learning methods make profiling and tracking users easier and faster than ever. Personal style and private information are in all types of textual data. End users need to be aware of how easily identifiable they are by their low level features to self regulate their information sharing behavior accordingly by the help of privacy enhancing technologies.

De-anonymization of authors and automated methods of behavioral analysis can be used to enhance security. These security enhancing methods can also be considered privacy infringing. Increased awareness of such security enhancing and privacy infringing methods lead to demand for counteracting privacy enhancing methods. The demand for privacy enhancing technologies, such as anonymization [76], sanitization, and other censorship evasion tools can be expected to increase as awareness of privacy threats increases. Effective development of tools and policies to increase privacy are possible by first and foremost identifying these threats. As technology evolves, new privacy and security threats emerge that require more sophisticated methods for detection and evasion.

1.1 Statement of Thesis

This thesis argues the following statement:

*Style and privacy behavior expressed in language can be quantified and characterized.*
Fingerprints in natural and programming languages can be numerically represented by stylometric analysis of textual data. Privacy behavior can be extracted from contextual properties of language. Such quantified stylistic fingerprints and behavioral representations can be used to characterize human behavior.

1.2 Key Contributions

1. Information about individuals can be extracted or inferred both from high level content that is consciously shared and low level linguistic properties of text that are unconsciously revealed.

   • Authors of anonymous text can be predicted even from challenging data, such as translated text, micro-text, a mixture of languages, and also source code.

   • Identifying authors in social networks can be used to link the accounts of same users within a network or across different networks.

   • In the case of source code, software forensics, copyright disputes, and plagiarism investigations can be resolved more effectively with stylometric proof.

   • Code stylometry provides software engineering insights such as how programming style changes while implementing sophisticated functionality or the differences in coding styles of programmers with different skill sets.

2. High level properties of text such as topics and named entities along with sentiment analysis makes it possible to quantify private information to associate a privacy score with each user.
Privacy behavior in social networks can be analyzed through privacy scores and preliminary results show that privacy is a collective behavior.

All kinds of textual data contain user fingerprints and create an online persona for each Internet user. Being able to identify these fingerprints helps an end user answer three main questions.

- What data do I consider sensitive?
- In what contexts should I share sensitive data?
- What does my data say about me?

1.3 Thesis Organization

In chapter 2 I discuss how de-anonymizing authors is possible through creating numeric representations of their writing style by extracting low-level linguistic and grammatical features. The media [40; 43; 87; 86; 31; 58] have used this work to raise awareness on online privacy. In section 2.1 I discuss how authors are de-anonymized in cyber criminal forums, where each author may use multiple aliases and communicate with text composed of product information, slang, and multiple languages. Such methods help experts in forensics detect abusive account holders or cyber criminals, as well as by providing insights about the forums. On the other hand, these methods pose a threat to privacy and as demonstrated in section 2.2 may expose users of multiple aliases aiming to evade censorship.

Translating text has been suggested as an anonymization method to rid a text of stylistic fingerprints. In section 2.3 I discuss that machine translation does not prevent an author from being identified [27]. I have also researched the portability of stylometric techniques used on natural languages to programming languages. Plain source code does not directly reflect the grammar of a program, however, parsing the code and generating its abstract syntax tree reveals the structure and functionality of
In section 2.4 I discuss how programmers can be de-anonymized by extracting syntactic information from abstract syntax trees along with linguistic information from source code to generate numeric representations of their coding style [28]. Being able to de-anonymize programmers from their coding style can aid in forensics and resolving copyright disputes. Studying coding style can have applications in software engineering as well.

In chapter 3 I discuss how features extracted from text can be used to study the behavioral fingerprints of users [29]. Features related to privacy, such as named entities, topic modeling, Brown clustering and semantic classification can categorize the privacy behavior of a user in a social network. The combination of these natural language processing based features gives an overall picture of users’ privacy behavior and can be used for fine-grained analysis of causal effects behind information disclosure and network phenomena.

In chapter 4 I discuss how influencing factors behind privacy behavior can be used to design effective privacy nudges to encourage privacy preserving behavior in online social networks. These factors could have a significant role in designing effective privacy policies and educational initiatives. Alternatively, automated sanitization approaches can be developed to preserve privacy. There are no known automated approaches to sanitizing documents to prevent inferences of sensitive information. Automated sanitization can redact classified documents, prevent data loss in companies, and make it possible to distribute communication data of users’ to linguists, psychologists, sociologists without revealing personal information.
2. Linguistic Style Feature Extraction

Stylometry is a field that relies on linguistic information found in a document to perform authorship attribution. Stylometry is currently used in intelligence analysis and forensics. The 2009 Technology Assessment for the State of the Art Biometrics Excellence Roadmap (SABER) commissioned by the FBI \cite{116} stated that, “As non-handwritten communications become more prevalent, such as blogging, text messaging and emails, there is a growing need to identify writers not by their written script, but by analysis of the typed content.” Authorship attribution is the problem of determining a text’s author, which can be accomplished by stylometric analysis.

Even basic stylometry systems reach high accuracy in classifying authors correctly \cite{6}. Stylometric analysis becomes more challenging when training and testing data starts differing from formal English prose. A simple modification to text is translating it to another language and then back to English. Stylometry on translated text will be discussed in section 2.3. Authorship attribution becomes even more challenging in underground forums discussed in 2.1, where the input is micro-text that includes slang, unstructured sentences, mixture of different languages. Another common question about stylometry is if it can be applied to source code, which is a form of structured text, since it is possible to use stylometry on many forms of text in various natural languages. Section 2.4 shows that code stylometry is possible through syntactic, lexical and layout features, leading to 94% correct authorship attribution accuracy among 1,600 programmers.

The methods in the following subsections of feature extraction have been established by engineering a feature set specific to the problem in hand. They all use stylistic and linguistic features, either specific to translated text, informal cyber criminal text, or a programming language.
2.1 Author Identification and Multiple Identity Detection in Cyber Criminal Forums

This work was completed by Sadia Afroz with support from Aylin Caliskan-Islam, Ariel Stolerman, and Damon McCoy [10]. The text in section 2.1 excluding section 2.2.7, is from the Doppelgänger Finder paper [10]. Parts of text from the paper [10] was used in Sadia Afroz’s thesis [7] as well as this thesis.

Forum databases of underground cyber criminals have been leaked publicly. Analyzing these forums revealed the struggle forum administrators had because of abusive forum members that create multiple accounts even after they were banned. These multiple identities of one user, so called doppelgängers, posed a challenge for authorship attribution as well. Stylometric analysis in underground forums is already challenging because of the nature of the messages that contain multiple languages (German, English, Russian, Turkish), product information (stolen credit card information), l33t-sp34k (leet-speak), slang, and micro-text. On the other hand, doppelgängers pollute ground truth by introducing new and artificial classes to machine learning, whereas all accounts of a doppelgänger should belong to one class. Besides reaching state-of-the-art authorship attribution accuracy on this data set, I helped develop and evaluate Doppelgänger Finder [10], that is currently used by FBI to find alternate egos of individuals by their writing style. Removing product information and using a language independent feature set that handles l33t-sp34k made this approach possible. Sadia Afroz came up with the Doppelgänger Finder algorithm. I performed the manual analysis on the results of the semi-supervised tool which revealed that 12 previously unknown multiple identities were discovered among 221 members. The manual analysis also revealed insights about how cybercriminals make use of multiple identities.
Stylometric techniques require 5,000 words of text from a known author to generate an accurate model of her writing style. This limitation eliminated many of the users in underground forums from the stylometric analysis, since they had less than 5,000 words of text. To overcome this limitation, an approach consisting of representations of words as numeric vectors has been used to complement Doppelgänger Finder and automatically extracts personal information about forum users. Having this approach work on such a challenging domain is a promising result for all other common domains that are more natural language processing friendly, such as blogs, online social networks, and emails. Law enforcement would be able to use Doppelgänger Finder with the added extension to investigate cyber criminals or abusive accounts in a larger population with the guidance of the automatically extracted identifying information.

Underground forums are used as a rendezvous location for cybercriminals and play a crucial role in increasing efficiency and promoting innovation in the cybercrime ecosystem. These forums are frequently used by cybercriminals around the world to establish trade relationships and facilitate the exchange of illicit goods and services such as the sale of stolen credit card numbers, compromised hosts, and online credential theft. Linking different aliases to the same individual across sources of data to increase knowledge of a cybercriminal’s activities is a powerful ability. An anecdotal example of this analysis performed manually is the case of the Rustock botnet operator where his accounts were manually linked together from multiple leaked data sources including underground forum posts [102]. All this information provides valuable insights, about how much he was earning, who else he was dealing with, which paints a fairly rich picture of a botnet operator’s role in the underground cyber ecosystem.

Other information gleaned from underground forums is providing security re-
searchers, law enforcement, and policy makers valuable information on how the market is segmented and specialized, the social dynamics of the community, and potential bottlenecks that are vulnerable to interventions. These advances have been accomplished primarily through analysis of limited structured metadata and painstaking manual analysis. Because of the size of the data sets and the labor intensity of the task, there are limitations to what can be accomplished by these techniques.

In fiction and folklore, a doppelgänger is an apparition or double of a living person. Many underground forums use the word *doppelgänger* to refer to a duplicate account of a user in the forum. The use of doppelgängers is forbidden in these forums because it undermines the fragile trust between pseudonymous users engaged in risky, illegal behavior and enables them to take advantage of each other. Users suspected of using multiple accounts are commonly banned. Understanding how and why users persist in maintaining multiple identities can help identify the dynamics of trust relationships in these forums. Detecting doppelgängers is possible through stylometric analysis and provides insights about the nature of underground forums.

Linguistic analysis has recently been applied successfully to many security problems from using stylometry to identify anonymous bloggers [80], to using topic modeling to find job postings for web service abuse [60]. However, the underground forums present a particular challenge for text analytic techniques. The messages are short and tend to mix conversations with “products” such as credit card and bank account numbers, URLs, IP addresses, etc. Furthermore, the forums are written in a multilingual 133t-speak slang that renders most natural language processing tools such as part-of-speech taggers inaccurate. This language is often intentionally difficult to parse and speak even for native human speakers and serves to weed out outsiders. As such they represent a stress test of sorts for these approaches.
Key contributions are:

1. **Adapting authorship attribution to underground forums.** Author attribution is useful in the scenario where an analyst has an unknown piece of text and wishes to attribute it to one out of a set of suspects. This scenario may be useful in underground analysis on its own, but it is also a subroutine in the multiple account detection algorithm.

   Although some language-agnostic authorship attribution methods are available [59, 61] for this task, most of the highly accurate attribution methods [80, 6] are language specific for standard English. By using language-specific function words and parts-of-speech taggers, authorship attribution method provides high accuracy even with over 1,000 authors in difficult, foreign language texts. Authorship attribution in underground forums is possible with a feature set that incorporates the informal language, such as l33tsp34k, used in underground forums and data preprocessing methods that can remove non-conversational products from messages. These as a whole improve the accuracy by 10-15% beyond current state-of-the-art methods directly applied to underground forums.

2. **A general multiple author detection algorithm.** Unlike standard authorship attribution, identifying doppelgängers is an unsupervised learning problem and requires novel methods where all pairs of accounts are compared against each other. Existing methods for this problem [14, 96] based on distance have been evaluated by artificially splitting authors into multiple identities. These methods have reduced accuracy when applied to actual separate accounts, such as multiple blogs by the same author and improved methods are needed. Non-textual methods used to identify fraud or spam accounts are insufficient because they do not catch the high-value alternate identities used in these forums. **Doppelgänger Finder** evaluates all pairs of a set of authors for duplicate identities and returns a list of potential pairs, ordered
by probability. This list can be used by a forum analyst to quickly identify interesting multiple identities. *Doppelgänger Finder* has been validated on real-world blogs using multiple separate blogs per author and using multiple accounts of members in different underground forums.

3. **A practical manual analysis of an underground forum to identify previously unknown multiple identities.** Discovering and grouping unknown identities in cases when ground truth data is unavailable is possible by using *Doppelgänger Finder*.

Running *Doppelgänger Finder* on a German underground forum called Carders automatically revealed at least 10 new author pairs (and an additional 3 probable pairs) which would have been hard to discover without time consuming manual analysis. These pairs are typically high value identities. One user was creating such identities for sale to other users on the forum. Manual analysis provides insights on how and why these identities are created by these users and the purposes they serve in the forums.

2.1.1 Related Work

Underground Markets

Most of the past research on the underground market has focused on either analyzing structured metadata (i.e. social graphs, and trade ratings) in underground forums or performing a manual analysis of products and prices. One of the first studies by Franklin et al. performed an analysis of underground chat messages in public IRC channels to gain insight into prices and types of products traded [45]. Another study performed an analysis of an underground carders forums to understand how they propagate credentials in large scale data breaches [91]. A separate study explored how trust models were formed in underground forums [79].
al. preformed an analysis of structural metadata in underground forums to examine the dynamics of social graphs in these communities [121]. Finally, another study did an analysis of activities taking place on Chinese underground markets [123]. McCoy et al. [75] analyzed the underground forums of three pharmaceutical affiliate programs and provided a detailed cost accounting of the overall business model. Recent research has investigated using underground market data to disrupt fraudulent activities. Thomas et al. identified patterns in fraudulent account usernames/emails by purchasing twitter accounts from an underground market [112].

When one forum is disrupted, these cybercriminals often create or join another forum using the same or different identity. Previous research tried to understand why these cybercriminals choose forums for doing their business [122] and what properties make underground forums sustainable [11]. Doppelgänger Finder focuses on detecting multiple accounts that are controlled by the same person based on automated analysis of the unstructured message contents. This approach can help identify known cybercriminals by analyzing their conversation, even when they change online identities.

Authorship Attribution

Users are unique in many ways and an extensive amount of research exploits different aspects of behavior to de-anonymize users in anonymized data sets. For example, a user can be identified based on how and what he types [30], his browser setup [41], which movie he prefers [82], who he connected with in a social network [83], when and what he writes in his blog or social network or on product reviews [52] [80] [15] and even how he fills bubbles in a paper form [26]. In the leaked underground forum, we only have the users’ posts and their social network information. But de-anonymizing these users using their social links from other social networks [83] is
challenging as these relationships are ephemeral business relationships. Also, often these posts are from different time frames, so linking users using timing analysis, as previous work did to de-anonymize flickr and twitter users is not possible [52] [82].

While stylometry has been applied to chat data in the past [6], large numbers of authors [80], as well as foreign language, and translated texts [27], the combination of these properties in this data set is unique. The Writeprints [6] work evaluated their techniques on instant messaging chat logs from CyberWatch (www.cyberwatch.com). This data set is probably the closest to the forum data sets. However, they had fewer words per author (an average of 1,422 words), but were in English. In this work, the accuracy is better even though there are more authors.

Some previous work has explored the question of identifying multiple identities of an author. The Writeprints method can be used to detect similarity between two authors by measuring distance between their “writeprints.” Qian et al.’s method, called “learning by similarity,” learns in the similarity space by creating a training set of similar and dissimilar documents [96] and comparing the distances between them. This method was evaluated using users on Amazon book reviews. Almishari et al. [14] also used a similar distance-based approach using reviews from yelp.com to find duplicate authors. Koppel et al. [63] used a feature subsampling approach to detect whether two documents are written by the same author. But all of these methods were evaluated by creating artificial multiple identities per author by splitting a single author into two parts.

Detecting Fraudulent Accounts

Perito et al. [92] showed that most users use similar usernames for their accounts in different sites, e.g., daniele.perito and d.perito. Thus different accounts of a user can be tracked by just using usernames. This does not hold when the users are
deliberately trying to hide their identity, which is often the case in underground forums (example of usernames in multiple accounts are in Table 2.10). Usernames and other account information and behavior in the social network have often used to identify sybil/spam accounts [49; 38; 16]. Doppelgänger Finder has a different goal, as it tries to identify duplicate accounts of highly active users, who would be considered as honest users in previous fraud detection papers. For example, these doppelgänger users are highly connected with other users in the forum, unlike spam/sybil accounts. Their account information (usernames, email addresses) are similar to spam accounts with mixed language, special characters, and disposable email accounts. However, these properties hold for most users in underground forums, even for the ones who are not creating multiple identities.

2.1.2 Overview of Underground Forums

SQL dumps of forum databases were available for the following underground forums: AntiChat (AC), BlackhatWorld (BW), Carders (CC), L33tCrew (LC) (summarized in Table 2.1). The complete SQL dumps of the databases include user registration information, along with public and private messages. Each of these SQL forum dumps has been publicly “leaked” and uploaded to public file downloading sites by unknown parties. Previous research performed on data collected by crawling or joining the forum. As a result, only the public portions of the forums were available for analysis. Leaked databases provide access to all the public and private messages in a specific time duration for each of the forums.

This section gives an overview of the forums, in particular, it shows the relationship between a member’s rank and his activities in the forum. In all forums, high-ranked members had more posts than low-ranked members. Access to special sections of these forums depends on a member’s rank. Having the full SQL dump gives the advantage
of seeing the whole forum, which would have been unavailable to an outsider or a newly joined member crawling the forum. In general, the high-ranked users have more reputation, a longer post history, and consequently more words for automated analysis.

Properties of Antichat

Antichat started in May 2002 and was leaked in June 2010. It is a predominantly Russian language forum with 25,871 active users (users with at least one post in the forum). Antichat covers a broad array of underground cybercrime topics such as password cracking, stolen online credentials, email spam, search engine optimization (SEO), and underground affiliate programs.

Anybody with a valid email address can join the forum, though access to certain sections of the forum is restricted based on a member’s rank. At the time of the leak, there were 8 advanced groups and 8 user ranks in the data set. A member of level N can access groups at level \( \leq N \). Admins and moderators have access to the whole forum and they grant access to levels 3 to 6 by invitation. At the time of the leak, there were 4 admins and 89 moderators in Antichat.

Members earn ranks based on their reputation which is given by other members of the forum for any post or activity. Initially each member is a Beginner (Новичок), a member with at least 50 reputation is Knowledgeable (Знаючи) and 888 reputation

---

2. Member rules are described [https://forum.antichat.ru/thread72984.html](https://forum.antichat.ru/thread72984.html)
3. Translated by Google translator
is a *Guru (Гуру)* (all user reputation levels are shown in Table 2.2). A member can also receive negative reputation points and can get banned. There were 3,033 banned members. The top reasons for banning a member are having multiple accounts and violating trade rules.

<table>
<thead>
<tr>
<th>Rank</th>
<th>Rep.</th>
<th>Members</th>
<th>Members with &gt;=4,500 words</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ламер (Lamer)</td>
<td>-50</td>
<td>646</td>
<td>22</td>
</tr>
<tr>
<td>Чайник (Newbie)</td>
<td>-3</td>
<td>340</td>
<td>4</td>
</tr>
<tr>
<td>Новичок (Beginner)</td>
<td>0</td>
<td>38,279</td>
<td>553</td>
</tr>
<tr>
<td>Знаво́цій (Knowledgeable)</td>
<td>50</td>
<td>595</td>
<td>256</td>
</tr>
<tr>
<td>Специалист (Specialist)</td>
<td>100</td>
<td>658</td>
<td>413</td>
</tr>
<tr>
<td>Эксперт (Expert)</td>
<td>350</td>
<td>271</td>
<td>177</td>
</tr>
<tr>
<td>Гу́ру (Guru)</td>
<td>888</td>
<td>206</td>
<td>153</td>
</tr>
<tr>
<td>Античатовец (Antichatian)</td>
<td>5,555</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 2.2: AntiChat Members Rank

Antichat has a designated “Buy, Sell, Exchange” forum for trading. Most of the transactions are in WebMoney[^4]. To minimize cheating, Antichat has paid “Guarantors” to guarantee product and service quality[^5]. Sellers pay a percentage of the value of one unit of goods/services to the guarantor to verify his product quality. Members are advised not to buy non-guaranteed products. In case of a cheating, a buyer is paid off from the guarantor’s collateral value.

[^4]: http://www.wmtransfer.com/
[^5]: https://forum.antichat.ru/thread63165.html
Properties of BlackhatWorld

BlackhatWorld is primarily an English speaking forum that focuses on blackhat search engine optimization techniques. It started in October 2005 and is still active. At the time of the leak (May 2008), Blackhat had 4,489 active members.

Like Antichat, anybody can join the forum and read most of the public posts. At the time of the leak, a member needed to pay $25 to post in a public thread. A member can have 8 ranks depending on his posting activities and different rights in the forums based on his rank. This rank can be achieved either by being active in the forum for a long period or by paying fees. A new member with less than 40 posts is a Blacknoob and 40-100 posts is a Peasant, both of these ranks do not have access to the “Junior VIP” section of the forum which requires at least 100 posts. The “Junior VIP” section is not indexed by any search engines or visible to any non Jr. VIP members. At the time of the leak, a member could pay $15 to the admin to access this section. A member is considered active after at least 40 posts and 21 days after joining the forum. Member ranks are shown in Table 2.3. The forum also maintains an “Executive VIP” section where membership is by invitation and a “Shitlist” for members with bad reputations. There were 43 banned members in our data set. Most of the members in the BlackhatWorld data set were Blacknoobs.

Currently, only the Junior VIP members can post in the BlackhatWorld marketplace, the “Buy, Sell, Trade” section. Any member with over 40 posts was allowed to trade. Each post in the marketplace must be approved by an admin or moderator. In the data set, there were 3 admins and 5 moderators. The major currency of this forum is USD. Paypal and exchange of products are also accepted.

---

6 The posting cost is now $30.
<table>
<thead>
<tr>
<th>Rank</th>
<th>Members</th>
<th>Members with &gt;=4,500 words</th>
</tr>
</thead>
<tbody>
<tr>
<td>Banned Users</td>
<td>43</td>
<td>4</td>
</tr>
<tr>
<td>21 days 40 posts</td>
<td>7,416</td>
<td>4</td>
</tr>
<tr>
<td>Registered Member</td>
<td>248</td>
<td>74</td>
</tr>
<tr>
<td>Exclusive V.I.Ps</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Premium Members (PAID/Donated)</td>
<td>191</td>
<td>19</td>
</tr>
<tr>
<td>Admins and Moderators</td>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>

Table 2.3: Blackhat Members Rank

Properties of Carders

Carders was a German language forum that specialized in stolen credit cards and other accounts. This forum was started in February 2009 and was leaked and closed in December 2010.

At the time of the leak, Carders had 3 admins and 11 moderators. A regular member can have 9 ranks, but unlike other forums the rank was not dependent only on the number of posts (Table 2.4). Access to different sections of the forum was restricted based on rank. Any member with a verified email can be a Newbie. A member needs at least 50 posts to be a Full Member. A member had to be at least a Full Member to sell tutorials. VIP Members were invited by other high-ranked members. To sell products continuously, a member needs a Verified vendor license which requires at least 50 posts in the forum and a fee of €150+ per month. For certain products, for example, drugs and weapons, the license costs at least €200. Carders maintained a “Ripper” thread where any member can report a dishonest trader. A suspected ripper was assigned Ripper-Verdacht! title. Misbehaving members, for example, spammers, rippers, or members with multiple accounts, were either banned temporarily or per-

\[^9\]Details of carders leak at [http://www.exploit-db.com/papers/15823/](http://www.exploit-db.com/papers/15823/)
manently depending on the severity of their action. In the data set, there were 1,849 banned members. The majority of the members in the Carders data set are Newbie.

<table>
<thead>
<tr>
<th>Rank</th>
<th>Members</th>
<th>Members with $\geq 4,500$ words</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nicht registriert (Not registered)</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Email verification</td>
<td>323</td>
<td>1</td>
</tr>
<tr>
<td>Newbie</td>
<td>4,899</td>
<td>23</td>
</tr>
<tr>
<td>Full Member</td>
<td>1,296</td>
<td>431</td>
</tr>
<tr>
<td>VIP Member</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>Verified Vendor</td>
<td>16</td>
<td>6</td>
</tr>
<tr>
<td>Admins</td>
<td>14</td>
<td>13</td>
</tr>
<tr>
<td>Ripper-Verdacht! (Ripper suspected)</td>
<td>14</td>
<td>7</td>
</tr>
<tr>
<td>Time Banned</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>Perm Banned</td>
<td>1849</td>
<td>193</td>
</tr>
</tbody>
</table>

Table 2.4: Carders Members Rank

Other products traded in this forum were cardable shops (shops to monetize stolen cards), proxy servers, anonymous phone numbers, fake shipping and delivery services, and drugs. The major currencies of the forum were Ukash\textsuperscript{10}, PaySafeCard (PSC)\textsuperscript{11}, and WebMoney.

Properties of L33tCrew

Like Carders, L33tCrew was a predominantly carding forum. The forum was started in May 2007 and leaked and closed in Nov 2009. Many users joined Carders after L33tCrew was closed. At the time of the leak, L33tCrew had 9,528 active users.

L33tCrew member rank also depended on a member’s activity and number of

\textsuperscript{10}https://www.ukash.com/
\textsuperscript{11}https://www.paysafecard.com/
posts. A member with 15 posts was allowed in the base account area. The forum shoutbox, which was used to report minor problems or off-topic issues, is visible to members with at least 40 posts. A member’s ranking was based on his activity in the forum (Table 2.5). On top of that, a member could have 2nd and 3rd level rankings. 100–150 posts were needed to be a 2nd level member. Members could rise to 3rd level after “proving” themselves in 2nd level and proving that they had non-public tools, tricks, etc. A 2nd level member had to send at least three non-public tools to the admin or moderators to prove himself.

<table>
<thead>
<tr>
<th>Rank</th>
<th>Min. posts</th>
<th>Members</th>
<th>Members with &gt;=4,500 words</th>
</tr>
</thead>
<tbody>
<tr>
<td>Newbie</td>
<td>0-30</td>
<td>715</td>
<td>93</td>
</tr>
<tr>
<td>Half-Operator</td>
<td>60</td>
<td>158</td>
<td>67</td>
</tr>
<tr>
<td>Operator</td>
<td>100</td>
<td>177</td>
<td>121</td>
</tr>
<tr>
<td>Higher Levels</td>
<td>150</td>
<td>412</td>
<td>398</td>
</tr>
<tr>
<td>Unranked Members</td>
<td>–</td>
<td>16,482</td>
<td>679</td>
</tr>
<tr>
<td>Banned</td>
<td>–</td>
<td>847</td>
<td>197</td>
</tr>
<tr>
<td>Admins</td>
<td>–</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>Invited</td>
<td>–</td>
<td>33</td>
<td>8</td>
</tr>
<tr>
<td>Vorzeitig in der Handelszone</td>
<td>–</td>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 2.5: L33tCrew Members Rank

Member Overlap Between Forums

Common active users in the forums can be identified by matching their email addresses. Here “active” means users with at least one private or public message in a forum. Among the four forums, Carders and L33tCrew had 563 common users based on email addresses, among which 443 were active in Carders and 439 were active in
Identity Protection

In all of the forums, multiple identities were strictly prohibited. On Carders and Antichat, one of the main reasons for banning a member was creating multiple identities. Some users were taking measures to hide their identities. Several users were using disposable email addresses (562 in Carders, 364 in L33tCrew) from top well-known disposable email services, e.g., trashmail.com, owlpic.com, and 20minutemail.

Carders used an alternative-ego detection tool (AE detector)\[12\] which saves a cookie of history of ids that log into Carders. Whenever someone logs into more than one account, it sends an automated warning message to forum moderators saying that the forum has been accessed from multiple accounts. The AE detector also warns the corresponding members. Users who received warning messages from the AE detector were considered part of a multiple identity. There were 400 multiple identity groups formed by 1,692 members, where group size varied from 2 to 466 accounts (shown in Figure 2.1).

We suspect that the AE detector does not reflect multiple account holders perfectly. There are possible scenarios that would trigger the AE detector, e.g. when two members use a shared device to log into Carders or use a NAT/proxy. The corresponding users in these situations were considered as doppelgängers by the AE detector, which does not reflect the ground truth. Likewise, the AE detector may not catch all the alter egos, as some users may take alternate measures to log in from different sources. These suspicions were supported by the stylometric and manual analyses of Carders’ posts.

\[12\]http://www.vbulletin.org/forum/showthread.php?t=107566
Figure 2.1: Duplicate Account Groups Within Carders as Identified by the AE Detector. (Each dot is one user. There is an edge between two users if the AE detector considered them as duplicate users.)
Public and Private Messages

In a forum a member can send public messages to public threads and private messages to other members. In our data set we had both the public and private messages of all the members. Public messages are used to advertise/request products or services. In general, public messages are short and often have specific formats. For example, Carders specifies a specific format for public thread titles.

Private messages are used for discussing details of the products and negotiating prices. Sometimes members use their other email, ICQ, or Jabber addresses for finalizing trades.

2.1.3 Authorship Attribution

The goal in this section is to see how well stylometry works in the challenging setting of underground forums and how to adapt stylometric methods to improve performance.

Approach

We consider a supervised authorship attribution problem that given a document D and a set of authors $\mathcal{A} = \{A_1, ..., A_n\}$ determines who among the authors in $\mathcal{A}$ wrote $D$. The authorship attribution algorithm has two steps: training and testing. During training, the algorithm trains a classifier using $F$ features extracted from the sample documents of the authors in $\mathcal{A}$. In the testing step, it extracts features predefined in $F$ from $D$ and determines the probability of each author in $\mathcal{A}$ of being the author of $D$. It considers an author $A_{\text{max}}$ to be the author of $D$ if the probability of $A_{\text{max}}$ being the author of $D$, $Pr(A_{\text{max}} \text{ wrote } D)$, is the highest among all $Pr(A_i \text{ wrote } D), i = 1, 2, ... n$.

$k$-attribution is the relaxed version of authorship attribution that outputs $k$
top authors, ranked by their corresponding probabilities, \( Pr(A_i \ wrote \ D) \), where \( i = 1, 2, \ldots, k \) and \( k \leq n \).

**Feature Extraction**

The feature set contains lexical, syntactic, and domain specific features. The lexical features include frequency of n-grams, punctuation, and special characters. The syntactic features include frequency of language-specific parts-of-speech and function words. In our data set we used English, German, and Russian parts-of-speech taggers and corresponding function words. For English and German parts-of-speech tagging we used the Stanford log-linear parts-of-speech tagger \[113\]. For Russian parts-of-speech tagging we used TreeTagger \[104\] with Russian parameters\[13\]. Function words or stop words are words with little lexical meaning that serve to express grammatical relationships with other words within the sentence, for example, in English function words are prepositions (to, from, for), and conjunctions (and, but, or). We used German and Russian stop words from Ranks.nl (http://www.ranks.nl/resources/stopwords.html) as function words. Similar feature sets have been used before in authorship analysis on English text \[6; 80; 76\]. We modified the feature set for the multilingual case by adding language specific features. As the majority of the members use leetspeak in these forums, we used the percentage of leetspeak per document as a feature. Leetspeak (also known as Internet slang) uses combinations of ASCII characters to replace Latin letters, for example, leet is spelled as l33t or 1337. We defined leetspeak as a word with symbols and numbers and used regular expressions to identify such words.

We used the JStylo \[70\] API for feature extraction, augmenting it with leetspeak percentage and the multilingual features for German and Russian.

\[13\]http://corpus.leeds.ac.uk/mocky/
<table>
<thead>
<tr>
<th>Feature</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Freq. of punctuation (e.g. &quot;,&quot;, &quot;.&quot;)</td>
<td>Dynamic</td>
</tr>
<tr>
<td>Freq. of special characters (e.g., '@', '%')</td>
<td>Dynamic</td>
</tr>
<tr>
<td>Freq. of character ngrams, n = 1-3</td>
<td>150</td>
</tr>
<tr>
<td>Length of words</td>
<td>Dynamic</td>
</tr>
<tr>
<td>Freq. of numbers ngrams, n = 1-3</td>
<td>110</td>
</tr>
<tr>
<td>Freq. of parts-of-speech ngrams, n = 1-3</td>
<td>150</td>
</tr>
<tr>
<td>Freq. of word ngrams, n = 1-3</td>
<td>150</td>
</tr>
<tr>
<td>Freq. of function words, e.g. for, to, the.</td>
<td>Dynamic</td>
</tr>
<tr>
<td>Percentage of leetspeak, e.g., l33t, pwn3d</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 2.6: Feature Set

Classification

We used a linear kernel Support Vector Machine (SVM) with Sequential Minimal Optimization (SMO) [94]. We performed 10-fold cross-validation, that is, our classifier was trained on 90% of the documents (at least 4,500 words per author) and tested on the remaining 10% of the documents (at least 500 words per author. This experiment is repeated 10 times, each time randomly taking one 500-word document per author for testing and the rest for training. To evaluate our method’s performance we use precision and recall. Here true positive for author A means number of times a document written by author A was correctly attributed to author A and false positive for author A means number of times a document written by any other author was attributed to author A. We calculate per author precision/recall and take the average to show overall performance.

Removing Product Data

One of the primary challenges with this data set is the mixing of conversational discussion with product discussions, e.g., stolen credentials, account information with
passwords, and exploit code. This is particularly pronounced in the most active users who represent the majority of the trading activities. As the classifier relies on writing style to determine authorship, it misclassifies when two or more members share similar kinds of product information in their messages. Removing product information from conversation improved the classifier’s performance by 10-15%. Identifying product information is also useful for understanding what kind of products are being traded in the forums.

Our product detector is based on two observations: 1) product information usually has repeated patterns, 2) conversation usually has verbs, but product information does not have verbs. To detect products, we first tag all the words in a document with their corresponding parts-of-speech and find sentence structures that are repeated more than a threshold of times. We consider the repeated patterns with no verbs as products and remove these from the documents.

To find repeated patterns, we measured Jaccard distance between each pair of tagged sentences. Due to errors in parts-of-speech tagging, sometimes two similar sentences are tagged with different parts-of-speech. To account for this, we considered two tagged sentences as similar if their distance is less than a threshold. We consider a post as a product post if any pattern is repeated more than three times. Note that our product detector is unsupervised and not specific to any particular kind of product, rather it depends on the structure of product information.

To evaluate our product detector we randomly chose 10,000 public posts from Carders and manually labeled them as product or conversation. 3.12% of the posts contained products. Using a matching threshold of 0.5 and repetition threshold of 3, we can detect 81.73% of the product posts (255 out of 312) with 2.5% false positive rate.

\[^{14}\text{Note that false positives are not that damaging, since they only result in additional text being removed.}\]
2.1.4 Results

Minimum Text Requirement for Authorship Attribution

![Figure 2.2: Effect of Number of Words Per User on Accuracy](image)

We trained our classifier with different numbers of training documents per author to see how much text is required to identify an author with sufficient accuracy. We performed this experiment for all the forums studied. In our experiments, accuracy increased as we trained the classifier with more words-per-author. On average, the accuracy did not improve when more than 4500 words-per-author were used in training (Figure 2.2).
Attribution Within Forums

Many users were removed from the data set due to insufficient text, especially after products and data dumps were removed. Table 2.7 shows the number of authors remaining in each forum and our results for author attribution in each forum which are mostly the high ranked members (section 2.1.2). Results are for the public and private messages respectively. Aside from this, performance on private messages ranged from 77.2% to 84% precision. Recall results were similar, as this is a multi-class rather than a binary decision problem and precision for all authors was averaged (a false positive for one author is a false negative for another author). This is comparable to results on less challenging stylometry problems, such as English language emails and essays [6]. Performance on public messages, which were shorter and less conversational—more like advertising copy—was worse, ranging from 60.3% to 72%. The product detection and changes to the features set we made increased the overall accuracy by 10-15% depending on the setting.

However, it is difficult to compare the performance across different forums due to the differing number of authors in each forum. Figure 2.3 shows the results of k-attribution for $k = 1$ to $k = 10$ where the $k = 1$ case is strict authorship attribution. In this figure we can see that the differences between private and public messages persist in this case and that the accuracy is not greatly affected when the number of authors scale from 50 to the numbers in Table 2.7. Furthermore, this figure shows that the results are best for the Carders forum. The higher accuracy for Carders and L33tCrew may be due to the more focused set of topics on these forums or possibly the German language. Via manual analysis, we noted that the part-of-speech tagger we used for Russian was particularly inaccurate on the Antichat data set. A more accurate part-of-speech tagger might lead to better results on Russian language forums.
Relaxed or k-attribution is helpful in the case where stylometry is used to narrow the set of authors in manual analysis. As we allow the algorithm to return up to 10 authors, we can increase the precision of results returned to 96% in the case of private messages and 90% in the case of public messages.

<table>
<thead>
<tr>
<th>Forum</th>
<th>Public</th>
<th>Private</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Members</td>
<td>Precision</td>
</tr>
<tr>
<td>AntiChat</td>
<td>1,459</td>
<td>44.4%</td>
</tr>
<tr>
<td>Blackhat</td>
<td>81</td>
<td>72%</td>
</tr>
<tr>
<td>Carders</td>
<td>346</td>
<td>60.3%</td>
</tr>
<tr>
<td>L33tCrew</td>
<td>1,215</td>
<td>68.8%</td>
</tr>
</tbody>
</table>

Table 2.7: Author Attribution Within a Forum.
Importance of Features

<table>
<thead>
<tr>
<th>German forums</th>
<th>English forums</th>
<th>Russian forums</th>
</tr>
</thead>
<tbody>
<tr>
<td>Char. trigram: mfg</td>
<td>Punctuation: ( )</td>
<td>Char. 1-gram: (ё )</td>
</tr>
<tr>
<td>Punctuation: Comma</td>
<td>Punctuation: Comma</td>
<td>Function word: емè (Trans.: more)</td>
</tr>
<tr>
<td>Leetspeak</td>
<td>Foreign words</td>
<td>Punctuation: Dot</td>
</tr>
<tr>
<td>Punctuation: Dot</td>
<td>Leetspeak</td>
<td>Char. 3-grams: ени</td>
</tr>
<tr>
<td>Char 3-gram: (...)</td>
<td>Function word: i’m</td>
<td>Char. bigrams: (, )</td>
</tr>
<tr>
<td>Nouns</td>
<td>Punctuation: Dot</td>
<td>Word-bigrams: что бы (that would)</td>
</tr>
<tr>
<td>Uppercase letters</td>
<td>POS-bigram (Noun,)</td>
<td></td>
</tr>
<tr>
<td>Function word: dass (that)</td>
<td>Char. bigram: (, )</td>
<td></td>
</tr>
<tr>
<td>Conjunctions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Char. 1-gram: ∧</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2.8: Features with Highest Information Gain Ratio in Different Forums

To understand which features were the most important to distinguish authors, we calculated the Information Gain Ratio (IGR) \[97] of each feature \( F_i \) over the entire data set:

\[
IGR(F_i) = \frac{H(A) - H(A|F_i))}{H(F_i)}
\]

(2.1)

where \( A \) is a random variable corresponding to an author and \( H \) is Shannon entropy.

Punctuation marks (comma, period, consecutive periods) were some of the most important features (shown in Table 2.8) in all the German, English, and Russian language forums. In German and English forums, leetspeak percentage was highly ranked. Interestingly, similar features are important across different forums, even though the predominant languages of the forums are different.

\(^{15}\)mfg is an abbreviation of a German greeting “Mit Freundlichen Gruessen” (English: sincerely yours).

\(^{16}\)German subordinating conjunctions (e.g. weil (because), daß (that), damit (so that))
2.2 Detecting Multiple Identities

In a practical scenario, an analyst may want to find any probable set of duplicate identities within a large pool of authors. Having multiple identities per author is not uncommon, e.g., many people on the Internet have multiple email addresses, accounts on different sites (e.g. Facebook, Twitter, G+) and blogs. Grouping multiple identities of an author is a powerful ability as the easiest way to change identity on the Internet is to create a new account.

Grouping all the identities of an author is not possible using only the traditional supervised authorship attribution. A supervised authorship attribution algorithm, trained on a set of unique authors, can answer who, among the training set, is the author of an unknown document. If the training set contains multiple identities of an author, supervised authorship attribution will identify only one of the identities as the most probable author, without saying anything about the connection among the authors in the training set.

2.2.1 Approach

Author identities can be grouped by leveraging supervised authorship attribution. For each pair of authors $A$ and $B$ we calculate the probability of $A$'s document being attributed to $B$ ($Pr(A \rightarrow B)$) and $B$’s document being attributed to $A$ ($Pr(B \rightarrow A)$). $A$ and $B$ are considered to be the same author if the combined probability is greater than a threshold. To calculate the pairwise probabilities, for each author $A_i \in \mathcal{A}$, train a model using all the other authors in $\mathcal{A}$ except $A_i$ and test using $A_i$. This method is called *Doppelgänger Finder*.

This method can be extended to larger groups. For example, for three authors $A$, $B$ and $C$, compute $P(A=\equiv B)$, $P(B=\equiv C)$, and $P(C=\equiv A)$. If $A=\equiv B$ and $C=\equiv B$, we consider $A$, $B$, and $C$ as the three identities of one author.
2.2.2 Feature Extraction

To identify similarity between two authors we use the same features as regular authorship attribution (Table 2.6), with two exceptions: 1) exclude the word n-grams, and 2) instead of limiting the number of other n-grams, use all possible n-grams. Word n-grams made the feature extraction process slower without any improvement in the performance. We used all possible character n-grams to increase the difference between authors, e.g., if author A uses a bi-gram “ng” but author B never uses it, then “ng” is an important feature to distinguish A and B. If we include all possible character n-grams instead of only the top 50, we can catch many such cases, specially the rare author-specific n-grams.

After extracting all the features, we add weight to the feature frequencies to increase distance among authors. This serves to increase the distance between present and not present features and gives better results. As our features contain all possible n-grams, the total number of features per data set becomes very large (over 100,000 for 100 authors). All the features are not important and they just make the classification task slower without improving the accuracy. To reduce the number of features without hurting performance, we use Principal Component Analysis (PCA) to weight and select only the features with high variance.

Principal component analysis (PCA) is a widely used mathematical tool for high dimension data analysis. It uses the dependencies between the variables to represent the data in a more tractable, lower-dimensional form. PCA finds the variances and coefficients of a feature matrix by finding the eigenvalues and eigenvectors. To perform PCA, the following steps are performed:

1. Calculate the covariance matrix of the feature matrix F. The covariance matrix measures how much the features vary from the mean with respect to each other.

   The covariance of two random variables X and Y is:
\[
\text{cov}(X, Y) = \frac{\sum_{i=1}^{N} (x_i - \bar{x})(y_i - \bar{y})}{N} 
\]  
\hspace{1cm} (2.2)

where \( \bar{x} = \text{mean}(X) \), \( \bar{y} = \text{mean}(Y) \) and \( N \) is the total number of documents.

2. Calculate eigenvectors and eigenvalues of the covariance matrix. The eigenvector with the highest eigenvalue is the most dominant principle component of the data set (PC1). It expresses the most significant relationship between the data dimensions. Principal components are calculated by multiplying each row of the eigenvectors with the sorted eigenvalues.

3. One of the reasons for using PCA is to reduce the number of features by finding the principal components of input data. The best low-dimensional space is defined as having the minimal error between the input data set and the PCA (eq. 2.3).

\[
\sum_{i=1}^{K} \lambda_i > \sum_{i=1}^{N} \lambda_i > \theta 
\]  
\hspace{1cm} (2.3)

where \( K \) is the selected dimension, \( N \) is the original dimension and \( \lambda \) is an eigenvalue. We chose \( \theta = 0.999 \) so that the error between the original data set and the projected data set is less than 0.1%.

2.2.3 Probability Score Calculation

We use logistic regression with ‘L1’ regularization with a regularization factor of \( C = 1 \) as a classifier to calculate pairwise probabilities. We experimented with linear kernel SVM, which was slower than logistic regression without any performance improvement. Any machine learning method that outputs classification probability scores can be used. After that we need to calculate \( P(A == B) \) by combining the
two probabilities: $P(A \rightarrow B)$ and $P(B \rightarrow A)$. We experimented with three ways of combining the probabilities:

1. Average: Given two probabilities $Pr(A \rightarrow B)$ and $Pr(B \rightarrow A)$, combined score is $\frac{Pr(A \rightarrow B) + Pr(B \rightarrow A)}{2}$.

2. Multiplication: Given two probabilities, combined score is $Pr(A \rightarrow B) \ast Pr(B \rightarrow A)$. We can consider the two probabilities as independent because when $Pr(A \rightarrow B)$ was calculated $A$ was not present in the training set. Similarly $B$ was not present when $Pr(B \rightarrow A)$ was calculated. Also in this case if any of the one-way probabilities are 0, the combined probability would be zero.

3. Squared average: The combined score is $\frac{Pr(A \rightarrow B)^2 + Pr(B \rightarrow A)^2}{2}$.

All the three approaches give similar precision/recall. We finally used the multiplication approach as its performance is slightly higher in the high recall region.

### 2.2.4 Multiple Identities in Underground Forums

In this section we show how Doppelgänger Finder method can be used to identify duplicate accounts by performing a case study on the underground forums. In the forums, many users create multiple identities to hide their original identity (reasons for doing so are discussed later) and they do so by changing the obvious identity indicators, e.g. usernames and email addresses. So we did not have any strong ground truth information for the multiple identities in a forum. We do, however, have some common users across two forums. We treat the common identities in multiple forums as one data set and use that to evaluate Doppelgänger Finder in underground forums. After that we run it on a forum and manually verify our results.
2.2.5 Multiple Identities Across Forums

We collected users with same email address from L33tCrew and Carders. We found 563 valid common email addresses between these two forums. Among them, 443 users were active (had at least one post) in Carders and 439 were active in L33tCrew. Out of these 882 users, 179 had over 4,500 words of text. We performed Doppelgänger Finder on these 179 authors which included 28 pairs of users (the rest of the 123 accounts did not have enough text in the other forum so merely served as distractor authors for the algorithm). Our method provides 0.85 precision and 0.82 recall when the threshold is 0.004 with exactly 4 false positive cases (Figure 2.4).

Figure 2.4: Doppelgänger Finder: With Common Users in Carders and L33tCrew: 179 Users with 28 Pairs. AUC is 0.82.
2.2.6 Multiple Identities Within a Forum

We used *Doppelgänger Finder* on Carders and manually analyzed the member-pairs with high scores to show that they are highly likely to be the same user. We selected all the Carders users with at least 4,500 words in their private messages, which resulted in a total of 221 users. We chose only private messages as our basic authorship attribution method was more accurate in private messages than in public messages. After that, we ranked the member pairs based on the scores generated by our method. The highest combined probability score of the possible pairs is 0.806 and then it goes down to almost zero after the first 50 pairs (Figure 2.5).

![Combined Probability Scores](image)

Figure 2.5: Combined Probability Scores of the Top 100 Pairs from Carders.
Table 2.9 shows the criteria we use to validate the possible doppelgängers. We manually read their private and public messages in the forum and information used in the user accounts to extract these features. The first criterion is to see if two users have the same ICQ numbers a.k.a UINs which is used by most traders to discuss details of their transactions. ICQ’s are generally exchanged in private messages. Our second criterion is to match signatures. In all the forums a user can enable or disable a default signature on their forum profiles. Signatures could be generic abbreviations of common phrases such as ‘mfg,’ or ‘Grüße’ or pseudonyms in the forum. We also investigate the products traded, payment methods used, topics of messages, and user information in the user table, e.g., join date, banned date if banned, rank in the forum and groups the user joined. We check whether or not they set off the Alter-Ego detector on Carders. Lastly we check whether or not members in a pair sent private messages to each other because that would indicate that they are likely not
the same person. We understand that there are many ways to verify identity but in most cases these serve as good indicators.

The Doppelgänger Finder algorithm considered \( \binom{221}{2} \) possible pairs. We chose all the pairs with score greater than 0.05 for our manual analysis (21 pairs). We limit our analysis to limit the number of pairs to analyze as it could be quite time consuming. We also chose three pairs with low score (rank 22-24 in Table 2.10) to illustrate that higher score pairs are more likely to be true match than the lower score pairs. Note that, all of the top possible doppelgängers use completely different usernames. To protect the members’ identity we only show the first three letters of their usernames in Table 2.10.

There are five possible outcomes of our manual analysis: True, Probably True, Unclear, Probably False and False. True indicates that we have conclusive evidence that the pair is doppelgängers, e.g., sometimes the pair themselves admit in their private/public messages about their other accounts or the pair shares same IM/payment accounts. Probably True indicates that the members share similar uncommon attributes but there’s no conclusive evidence of them being the same. Unclear indicates that some criteria are similar in both and some are very different and no conclusive attributes either way. Probably False means there are very few to no similarity between the members but no evidence that they are not the same. False indicates that we found conclusive evidence that the members in a pair are not the same, e.g., the members trade with each other.

**Results and Discussion**

We found that in Carders, the accounts produced at the high end of the probability range were doppelgängers. The 12 pairs with the highest probabilities were assessed as True or Probably True. After that, there is a range where both the manual
Table 2.10: Manual Analysis of Users: X indicates same, – indicates different, empty means the result is inconclusive or complicated with many values.

<table>
<thead>
<tr>
<th>Rank</th>
<th>Score</th>
<th>Usernames</th>
<th>ICQ</th>
<th>Sig.</th>
<th>Contact</th>
<th>Acc.</th>
<th>Topics</th>
<th>AE</th>
<th>Other</th>
<th>Intr.</th>
<th>Decision</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.806</td>
<td>per**, Smi**</td>
<td>X</td>
<td>icq</td>
<td>weed</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>True</td>
<td></td>
<td>True</td>
</tr>
<tr>
<td>2</td>
<td>0.799</td>
<td>Fri**, Loo**</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>True</td>
<td></td>
<td>True</td>
</tr>
<tr>
<td>3</td>
<td>0.601</td>
<td>Sch**, bob**</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>True</td>
<td></td>
<td>True</td>
</tr>
<tr>
<td>4</td>
<td>0.495</td>
<td>Duk**, Mer**</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>True</td>
<td></td>
<td>True</td>
</tr>
<tr>
<td>5</td>
<td>0.474</td>
<td>Dra**, Pum**</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>True</td>
<td></td>
<td>True</td>
</tr>
<tr>
<td>6</td>
<td>0.372</td>
<td>p01**, tol**</td>
<td></td>
<td>greez</td>
<td></td>
<td>X</td>
<td>0</td>
<td></td>
<td>Probably True</td>
<td></td>
<td>True</td>
</tr>
<tr>
<td>7</td>
<td>0.342</td>
<td>Qui**, gam**</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>True</td>
<td></td>
<td>True</td>
</tr>
<tr>
<td>8</td>
<td>0.250</td>
<td>Uni**, Ra**</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>True</td>
<td></td>
<td>True</td>
</tr>
<tr>
<td>9</td>
<td>0.196</td>
<td>PUN**, nec**</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>True</td>
<td></td>
<td>True</td>
</tr>
<tr>
<td>10</td>
<td>0.192</td>
<td>Koo**, Wec**</td>
<td></td>
<td>peace</td>
<td>weed</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>Probably False</td>
<td></td>
<td>True</td>
</tr>
<tr>
<td>11</td>
<td>0.176</td>
<td>To**, Have**</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>Probably False</td>
<td></td>
<td>True</td>
</tr>
<tr>
<td>12</td>
<td>0.140</td>
<td>Xer**, kike**</td>
<td>X</td>
<td></td>
<td>weed</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>Unsure</td>
<td></td>
<td>False</td>
</tr>
<tr>
<td>13</td>
<td>0.095</td>
<td>Xer**, pat**</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>Probably False</td>
<td></td>
<td>False</td>
</tr>
<tr>
<td>14</td>
<td>0.072</td>
<td>Qui**, Sco**</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>0</td>
<td></td>
<td>False</td>
<td></td>
<td>False</td>
</tr>
<tr>
<td>15</td>
<td>0.058</td>
<td>Het**, meo**</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>False</td>
<td></td>
<td>False</td>
</tr>
<tr>
<td>16</td>
<td>0.051</td>
<td>Mi**, Flo**</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>26</td>
<td>False*</td>
<td></td>
<td>False*</td>
</tr>
<tr>
<td>17</td>
<td>0.01</td>
<td>psi**, pot**</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>False</td>
<td></td>
<td>False</td>
</tr>
<tr>
<td>18</td>
<td>0.051</td>
<td>Bo**, Fru**</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>False</td>
<td></td>
<td>False</td>
</tr>
<tr>
<td>19</td>
<td>0.001</td>
<td>Car**, Din**</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>False</td>
<td></td>
<td>False</td>
</tr>
</tbody>
</table>

and linguistic evidence is thinner but nonetheless contains some true pairs (pairs 13-17). The manual analysis suggested that pairs below this probability threshold were likely not doppelgängers. Thus, our manual analysis overall agreed with the linguistic analysis performed by Doppelgänger Finder.

**True** True cases are particularly seen when users explicitly state their identities and/or use the same ICQ numbers in two separate accounts. For example, each pair of users in **Pair 1-3, 5, 6, 8 9, 10, and 16** provides an ICQ number in their private messages that is unique to that pair. The users in **Pair-11** use the same jabber nickname. One of the users in **Pair 1** (user name per**) was asking the admins to give his other account back and telling other members that he is Smi**.

Other cases had just as convincing, but more subtle evidence. The accounts in **Pair-8** both use trashmail which provides disposable email addresses, which shows that these users are careful about hiding their identities. However, the most convincing evidence of their connection was a third doppelgänger account, which we will call
user-8c, who did not have enough text to be in our initial user set, but was brought to our attention by the linguistic similarity between the accounts in Pair-8. Both users in Pair-8 share the same ICQ number with user-8c. User-8b explicitly writes two messages from User-8c’s account, one in Turkish and one in English revealing his user-8b username. These users do not send private messages to each other. These findings imply that the three user accounts belong to the same person.

**Probably True** These accounts do not have a “smoking gun” like a shared ICQ number or Jabber account, however, we are able to observe that the accounts shared have similar interests or other properties. We consider how common these similar properties are in the entire forum and assess as probably true accounts that share uncommon properties.

In the case of Pair-4, user-4a does not have an ICQ number, but user-4b frequently gives out an ICQ number. User-4a wants to buy new ICQ numbers. This suggests that he uses ICQ and hides his own ICQ number. They both use a similar signature: ‘mfg’, but this is common. They trade similar products and talk about similar topics such as Kokain and D2 numbers. Since these are not common, this suggests they might be the same user. User-4a is a newbie while user-4b is a full member. The accounts were active during the same period.

The accounts in Pair-7 have different ICQ numbers. However, both user-7a and user-7b deal with online banking products, PS3, Apple products, Amazon accounts and cards. They both use Ukash. They both use the same signature such as ‘grüße’ or ‘greezz’. User-7a is a full member and user-7b is permanently banned. They have both been active account holders at the same period. User-7a has a 13th level reputation and user-7b has a 11th level reputation.

Similarly, the accounts in Pair-12 use the same, rare signature ‘peace’ and both are interested in weed.
**Unclear** The accounts in **Pair-13** do not have common ICQ numbers, even though they have the same ICQ numbers with other users (suggesting they do use doppelgänger accounts with lower text, lower reputation accounts). User-13a is a full member with a reputation level of 8. User-13b is a full member with a reputation level of 15. User-13a’s products are carding, ps, packstation, netbook, camcorder, and user-13b’s products are carding, botnets, cc dumps, xbox, viagra, iPod.

**Probably False** The **Pair-14** accounts have different ICQs. User-14a products are tutorials, accounts, Nike, ebay and ps. User-14b’s products are cameras and cards. User-14a is a full member with reputation level of 5. User-14b is permanently banned with a reputation level of 15.

One of the users in **Pair-17**, User-17b shares two ICQ numbers with another user but not with User-17a. User-17a’s products are iPhone, iPad, macbook, drops, and paypal and User-17b’s products are: paypal, iPhone, D2 pins, and weed.

**False** These users have specific and different signatures and also they use different ICQ numbers. These accounts sometimes interact, suggesting separate identities.

Pairs such as **20** send each other private messages to trade and complete a transaction, suggesting they are business partners not doppelgängers.

The accounts in **Pair-24** do not have any common UINs. They have different signatures, User-24a uses the signature ‘LG Carlos’ and ‘Julix’ interchangeably. User-24b never uses ‘Carlos’ or ‘Julix’ but he sometimes uses ‘mfg’ or ‘DingDong’ at the end of his messages. User-24a’s products are iPhone, ebay, debit, iTunes cards, drop service, pack station, fake money while User-24b’s products are camera, ps3, paypal, cards, keys, eplus, games, perfumes. They do not talk to each other.

**Pair-21** is a special case of false labels. User-21a and user-21b belong to group accounts. User-21a tells user-21b: “*You think it is good that they think we are the*
same.", because they got a warning from the admins for using the same computer. In their private messages, they state that they are meeting at each other’s houses in person for business, which implies that they might be using the same accounts. They send many messages to other people mentioning each other’s names to customers.

2.2.7 Automating Forum Analysis

Stylometric methods require a certain amount of training data for accurate analysis. This limitation requires excluding forum users that have less than 5,000 words of messages from stylometric analysis. There are ways to overcome this limitation to perform more in-depth forum analysis, such as unsupervised language analysis methods that generate global vectors for word representations [90] and cluster the words according to semantic and syntactic similarity [77].

Generating word clusters by using the word2vec tool [78] leads to a quick understanding of the types of products that are being traded in the forums. Word2vec is a neural network that processes text without human intervention. Word2vec takes a string of sentences as its input and converts the words to n-dimensional vectors based on word co-occurrences. In this section, word vector representations have 50 dimensions. Given enough input with related words that appear in the same context, the similarity of these vectors can be measured and vectors can accordingly be clustered. These clusters can form the basis of search recommendations. The vectors can be used as features for many natural language processing and machine learning applications.

The word2vec implementation used throughout this section utilizes a continuous bag-of-words model with negative sampling. Word2vec was trained on the 11,127,050 words of public Carders messages that had 87,072 unique vocabulary words. For example, a large cluster (Table 2.11) out of 1,000 generated from public Carders
messages gives examples of some products, namely illicit drugs and the quantity associated with them. The clusters gives a quick understanding of the type of things that are of interest to this community as well as the extent of the businesses. On the other hand, the clusters generated from the public and private parts of the forums are quite similar. In cases of such similarity, gaining an understanding of a forum is possible through collecting only the public messages.

<table>
<thead>
<tr>
<th>0.5</th>
<th>1.5g</th>
<th>100G</th>
<th>100g</th>
<th>100g,</th>
<th>10g</th>
</tr>
</thead>
<tbody>
<tr>
<td>150g</td>
<td>1g</td>
<td>2.5g</td>
<td>200g</td>
<td>20g</td>
<td>25g</td>
</tr>
<tr>
<td>2g</td>
<td>300g</td>
<td>30g</td>
<td>4.5g</td>
<td>40g</td>
<td>50-100g</td>
</tr>
<tr>
<td>500g</td>
<td>50G</td>
<td>50g</td>
<td>5G</td>
<td>5g</td>
<td>5g.</td>
</tr>
<tr>
<td>6G</td>
<td>6g</td>
<td>Albi</td>
<td>Amnesia</td>
<td>Arjans</td>
<td>Buds</td>
</tr>
<tr>
<td>Chronic</td>
<td>DGC</td>
<td>Falschgeld</td>
<td>G</td>
<td>Gr.</td>
<td>Gra?</td>
</tr>
<tr>
<td>Gramm</td>
<td>Gras</td>
<td>Hasch</td>
<td>Hash</td>
<td>Haze</td>
<td>Heroin</td>
</tr>
<tr>
<td>Kostprobe</td>
<td>Kristalle</td>
<td>Kurs</td>
<td>Kurs.</td>
<td>MDMA</td>
<td>Material</td>
</tr>
<tr>
<td>Mephedron</td>
<td>Meth</td>
<td>Partypaket</td>
<td>Paste</td>
<td>Pep</td>
<td>Pepp</td>
</tr>
<tr>
<td>Probe</td>
<td>Probe,</td>
<td>Pulver</td>
<td>Quali</td>
<td>Rabbat</td>
<td>Ritalin</td>
</tr>
<tr>
<td>Salvia</td>
<td>Silver</td>
<td>Sorte</td>
<td>Speed</td>
<td>Speed,</td>
<td>Speedpaste</td>
</tr>
<tr>
<td>Stammkunde</td>
<td>Streckmittel</td>
<td>Test</td>
<td>Testbestellung</td>
<td>Testen</td>
<td>Testmenge</td>
</tr>
<tr>
<td>Testpacket</td>
<td>Top</td>
<td>Verk?ufen</td>
<td>Weed</td>
<td>Weed,</td>
<td>White</td>
</tr>
<tr>
<td>XTC</td>
<td>Zeug</td>
<td>faires</td>
<td>gestrecktes</td>
<td>gr</td>
<td>gr.</td>
</tr>
</tbody>
</table>

Table 2.11: A Product and Quantity Cluster from Public Carders Messages

Getting more detailed information about particular products is also possible. Instead of generating word clusters, the closest words to a particular product can be retrieved by calculating the cosine similarity between word vectors. Table 2.12 shows the results of querying the word ‘weed’ and getting the most related words in descending order of cosine distance.

Word vector representations can also be used to automate the manual evaluation of Doppelgänger Finder results. Word2vec was trained on the 5,604,188 private
<table>
<thead>
<tr>
<th>Word</th>
<th>Distance</th>
<th>Word</th>
<th>Distance</th>
<th>Word</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>zeug</td>
<td>0.575246</td>
<td>paste</td>
<td>0.562277</td>
<td>hasch</td>
<td>0.560486</td>
</tr>
<tr>
<td>speed</td>
<td>0.555898</td>
<td>pep</td>
<td>0.538496</td>
<td>pepp</td>
<td>0.523110</td>
</tr>
<tr>
<td>g</td>
<td>0.520100</td>
<td>gras</td>
<td>0.517192</td>
<td>kilo</td>
<td>0.512124</td>
</tr>
<tr>
<td>weed?</td>
<td>0.511901</td>
<td>haze</td>
<td>0.498430</td>
<td>gramm</td>
<td>0.483961</td>
</tr>
<tr>
<td>grass</td>
<td>0.477622</td>
<td>Weed</td>
<td>0.475446</td>
<td>Speed</td>
<td>0.468679</td>
</tr>
<tr>
<td>weed.</td>
<td>0.459948</td>
<td>mdma</td>
<td>0.458200</td>
<td>albi</td>
<td>0.457879</td>
</tr>
<tr>
<td>pillen</td>
<td>0.456605</td>
<td>kleinzeug</td>
<td>0.455731</td>
<td>10g</td>
<td>0.454204</td>
</tr>
<tr>
<td>100g</td>
<td>0.453324</td>
<td>weed,</td>
<td>0.453105</td>
<td>juden</td>
<td>0.451888</td>
</tr>
<tr>
<td>xtc</td>
<td>0.449392</td>
<td>testmenge</td>
<td>0.449025</td>
<td>25g</td>
<td>0.438363</td>
</tr>
<tr>
<td>kurs</td>
<td>0.434713</td>
<td>50g</td>
<td>0.434571</td>
<td>koka</td>
<td>0.432676</td>
</tr>
<tr>
<td>500g</td>
<td>0.428182</td>
<td>quali</td>
<td>0.421362</td>
<td>holland</td>
<td>0.418891</td>
</tr>
<tr>
<td>20g</td>
<td>0.414536</td>
<td>speed.</td>
<td>0.413150</td>
<td>sorte</td>
<td>0.407879</td>
</tr>
<tr>
<td>stoff</td>
<td>0.404382</td>
<td>hash</td>
<td>0.399687</td>
<td>probe</td>
<td>0.392878</td>
</tr>
</tbody>
</table>

Table 2.12: Words Closest to the Word: ‘weed’

Carders messages that had 38,625 vocabulary words. Querying the user pairs that Doppelgänger Finder outputs in order of probability is useful while performing manual analysis. This approach does not require the analyst to read all user messages and find user relations with sql queries. For example, in pair-2’s case, the user closest to user-2a is user-2b.

2.2.8 Lessons Learned about Underground Markets

Doppelgänger Finder helped us detect difficult to detect dopplegänger accounts. We performed a preliminary analysis on L33tCrew and Blackhat and found similar results as in Carders. Our manual analysis of these accounts improves our understanding of why people create multiple identities in underground forums, either within or across forums.
Reasons for Creating Multiple Identities

**Banning.** Getting banned in a forum is one of the main reasons for creating another account within a forum. Rippers, spammers or multiple account holders get penalized or banned once the admins become aware of their actions. Users with penalties get banned once their infraction points go over a certain threshold. There are hundreds of users within forums that have been banned and they open new accounts to keep actively participating in the forums. Some of the new accounts get banned again because the moderators realize that they have multiple accounts, which is a violation of forum rules.

**Sockpuppet.** Some forum members create multiple accounts in order to raise demand and start a competition to increase product prices.

**Accounts for sale.** Some users maintain multiple accounts and try to raise their reputation levels and associate certain accounts with particular products and customers. Once a certain reputation level is reached, they offer to sell these extra accounts.

**Branding.** Some users appear to setup multiple accounts to sell different types of goods. One reason to do this is if one class of goods is more risky, such as selling drugs, the person can be more careful about protecting their actual identity when using this account. Another reason to do this might be to have each account establish a “brand” that builds a good reputation selling a single class of goods, such as stolen credit cards.

**Cross-forum accounts.** Many users have accounts in more than one forum potentially as a method to grow in their sales by reaching more people not present on the same forums and to purchase goods not offered in a single forum.

**Group accounts.** In some cases groups of people work together as an organization and each member is responsible for a specific operation among a variety of products that are traded across different accounts. How to adapt stylometry algorithms to deal
with multi-authored documents is an open problem that is left as future work.

2.2.9 Lessons Learned about Stylometry

We found that any stylometric method can be used in a particular language by using a high quality parts-of-speech tagger and function words of that language. We have access to one more forum called BadhackerZ whose primary language is transliterated Hind using English letters. We did not have a POS tagger that could handle the mixture of these two languages. We were not able to get meaningful results by applying stylometry to BadhackerZ, therefore we excluded this forum from stylometric analysis. Similarly, the Russian POS tagger that was used produced poor results on our data set. POS tags generally have high information gain in stylometric analysis and as a result play a crucial role in stylometry. Future work might involve experimenting with other POS taggers or improving their efficacy by producing manually annotated samples of forum text.

2.2.10 Doppelgänger Detection by Forum Administrators

One of the primary reasons for banning accounts on these underground forums is because of users creating multiple accounts. This shows that forum administrators are actively looking for these types of accounts and removing them since they can be used to undermine underground forums. They use a number of methods ranging from automated tools, such as AE detector, and more manual methods, such as reports from other members. As we have seen from analysis all of these methods are error prone and result in many false positives and false negatives. Many of the false positives were probably generated by users using proxies to hide their IP and location. In addition, when static tools with defined heuristics (IPs, browser cookies, etc.) are used to detect doppelgänger accounts’ users can take simple precautions
to avoid detection. Many of the accounts detected by *Doppelgänger Finder* were not detected by these methods potentially because that user was actively evading known detection methods.

### 2.2.11 Performance

Our method needs to run N classifiers for N authors. Each classifier is independent, thus can be run in parallel. Using only 4 threads on a quad core Apple laptop, the underground forum *Doppelgänger Finder* experiment took around 35 minutes, which can be made faster with more threads.

### 2.2.12 Hybrid Doppelgänger Finder Methods

Based on what we have learned from our manual analysis of our *Doppelgänger Finder* results on Carders, we could potentially build a hybrid method that integrates both stylometry and more underground specific features. For instance, some of the doppelgänger accounts could be identified with simple regular expressions that find and match contact information, such as ICQ numbers. In other cases manual analysis revealed more subtle features, such as two accounts selling the same uncommon product or talk about a similar set of topics can be a good indicator that they are doppelgängers.

Custom parsers and pattern matchers could be created and combined with our *Doppelgänger Finder* tool to improve its results. However, it is difficult to know a priori what patterns to look for in different domains. Thus, using *Doppelgänger Finder* and performing manual analysis would make this task of designing and adding additional custom tools easier.
2.2.13 Methods to Evade Doppelgänger Finder

There are several limitations to using stylometry to detect doppelgängers. The most obvious limitation is that our method required a large number of words from a single account. A forum member could stop using their account and create a new one before reaching this amount of text, but as pointed out in Section 2.1.2 parts of the forum are closed off to new members, thus less activity is not beneficial to them. They are often not allowed to engage in commerce until they have payed a fee and built up a good reputation by posting.

Another way to evade our method is for the author to intentionally change their writing style to deceive stylometry algorithms. As shown in previous research this is a difficult, but possible task [21], and tools such as Anonymouth can give hints as to how to alter writing style to evade stylometry [76]. We do not currently see any evidence of this technique being used by members of underground forums, but Anonymouth could be integrated into forums.

2.2.14 Conclusion

Doppelgänger Finder enables easy analysis of a forum for high-value multiple identities. The analysis of Carders has already produced insight into the use of multiple identities within these forums. This technique can also be used to detect multiple identities on non-malicious platforms.

This work also motivates the need for improved privacy enhancing technologies such as Anonymouth [76] for authors who wish to not have their pseudonymous writings linked.
2.3 Author Identification in Translated Text

This work was completed by Aylin Caliskan-Islam. [27].

Internet scale authorship attribution is able to identify an individual 20% of the time among 100,000 authors. As the number of authors drops, the chances of being de-anonymized greatly increases. Someone trying to publish political opinions in an oppressive regime would not want to be identifiable, which calls for the need of anonymization techniques. “Anonymouth” is an anonymization framework based on JStylo, an authorship analysis tool, which was presented in a paper that won the PETS best student paper award [76]. I helped users anonymize their text by the use of “Anonymouth” while investigating the suggestions to translate text to anonymize it. Translations alone do not anonymize text [27]. A set of writing style features that are independent of language or not affected by translators still captures stylistic fingerprints. Translations from English to German to Japanese and back to English do not remove stylometric fingerprints from text as long as the machine translator has sufficient quality.

2.3.1 Introduction

Authorship attribution is the problem of determining a text’s author, which we can be accomplished by stylometric analysis. This is a serious privacy concern that prevents anonymous speech. Authorship attribution can still be achieved in translated texts using a set of features, indicating that the authors are not obfuscated.

Rao and Rohatgi [98] had introduced the idea of translating text to a different language and then back to its original language using a machine translation tool to obfuscate a text’s author. Translated text accumulates properties from the machine translation tool, which is called the translator effect. The translator effect introduces an extra author to the translated text, which is the machine translation tool itself.
A classifier can be trained to consider the machine translation tools’ footprints to attribute a translator to translated anonymous text. The translator effect does not prevent authorship attribution even though the translator introduces new features to the text.

Machine translators are categorized by the techniques they use to perform translations. Bing’s\textsuperscript{17} and Google’s\textsuperscript{18} translators both rely on statistical machine translation. When two translators use the same technique, as is the case with Bing’s and Google’s translators’ statistical machine translation, they do not produce the same output given the same input. Because of these differing translator effects, certain features can be used to identify the translator that has been used.

### 2.3.2 Related Work

State-of-the-art stylometry methods can identify individuals in sets of 50 authors with over 90% accuracy as shown in Abbasi and Chen’s work \cite{6}. There has not been much research on identifying the translator effect, translators, and authors in translated text. Suresh et al. \cite{110} were able to match the translated text with the machine translation tool used in the translation of the original text.

Hedegaard and Simonsen \cite{57} researched authorship attribution in translated text, which is outperformed in this work. They used classifiers based on frame semantics in order to discover whether adding semantic features to lexical and syntactic features would improve authorship attribution. Their studies were conducted on a corpus that had a limited number of authors from a specific time period and cultural context, which had only undergone a one-way translation.

\textsuperscript{17}http://www.microsofttranslator.com/
\textsuperscript{18}http://translate.google.com/
2.3.3 Corpus Selection

Data selection for authorship attribution is an important step. Luyckx and Daelemans [71] show that the number of authors and the amount of text have a big impact on the efficiency of classification. Brennan-Greenstadt Adversarial Stylometry Corpus [22] has thirteen authors and a minimum of 5,000 words per author. All writing samples are written in English by native English speakers. The adversarial stylometry corpus includes one obfuscation and one imitation document per author besides the author’s original writing. This corpus is used throughout the translated text section, after excluding the adversarial documents in order to experiment only with the original writing styles. The used parts of the corpus had thirteen authors, 126 documents containing an average of 4,933 words per author and 500 words per document. Forsyth and Holmes [44] show that a minimum of 250 words is required in text for authorship attribution. While testing authorship attribution accuracy on a range of data sets with documents varying from 400 to 600 words, 500 words per document led to highest accuracy. Accordingly, test documents consisted of 500 word chunks. The writing samples in the corpus have random topics and therefore are not content-dependent. Schein and Caver [103] show that attribution markers are influenced heavily by topics and effect the authorship attribution rate. Including varying topics among texts and authors avoids this effect.

In order to create the machine-translated texts, Bing’s and Google’s translators applied three different sequences of translations to the original corpus. The first sequence translated the original texts to German and then back to English. The second sequence translated the original texts to Japanese and then back to English. The third and last sequence translated the original texts to Japanese, then to German, and then back to English. Hedegaard and Simonsen used eighteen documents in their translator attribution experiments. Their corpus consists of English transla-
tions of 19th century Russian romantic literature. The experimented texts have three authors and three translators whereas this work has thirteen authors, two statistical machine translators and two or three consequent translations. The translated Brennan-Greenstadt Adversarial Stylometry Corpus has more translator effect in the translated text due to two or three consequent translations compared to their single translation from Russian to English. Additionally, Brennan-Greenstadt Adversarial Stylometry Corpus consists of modern text of diverse topics written in the 21st century. As a result, it is more diverse and current.

After performing the two-way translation experiments, feature set validation is performed on one-way translations. One-way translations consisted of the work of two French authors and four Dutch authors from the Ad-hoc Authorship Attribution Competition data set. These texts were translated to English by using Google Translate, Language Weaver and Systran. Both Language Weaver and Systran are also statistical machine translation tools. Each Dutch document ranged from 400 to 600 words. The Dutch data set consisted of essays on the same six topics by the four authors and is therefore topic-dependent. Because of these qualities in the Dutch data set, the effect of document length and topic-dependency on authorship and translator attribution can be observed.

French and Dutch belong to different language families, namely Romance and Germanic, and therefore possess different grammatical structures. This distinction between the two languages gives the opportunity to compare language family independent features in translator and authorship attribution.

19http://www.mathcs.duq.edu/juola/authorship_materials2.html
2.3.4 Experiment Design

The experiments had two main categories; namely, translator attribution and authorship attribution. Accuracy obtained from a variety of feature sets were compared to identify the features leading to the highest accuracy. The experiments utilized two authorship attribution tools, (1) JGAAP\(^\text{20}\) developed by Juola et al. \cite{59} and (2) JStylo, a novel framework for authorship attribution that was developed by McDonald et al. \cite{87}. JGAAP is limited to analysis using one feature at a time. The majority of experiments used JStylo, which is capable of using a set of multiple features.

Authorship attribution or translator attribution is a supervised authorship attribution problem which relies on correct ground truth authorship or translator information, that given a document \(D\), and its translation with translator \(T\) to another language and then back to English \(D^*\) and a set of unique authors \(A = \{A_1, ..., A_n\}\), where \(A_i \neq A_j\) when \(i \neq j\), determines who among the authors in \(A\) wrote \(D^*\), and which translator \(T\) was used.

Translator Attribution Experiments

The documents in the corpus were preprocessed and normalized by stripping all non-ASCII and non-printing characters while preserving the whitespace. Two machine learning classifiers were trained using JGAAP, namely a Naïve Bayesian classifier and a support vector machine with sequential minimal optimization (SMO) based on Platt’s \cite{94} method. The classifiers trained on features such as character grams, part-of-speech (POS) tags, word grams, word lengths, words, function words, sentence length, and rare words. The features with the most frequent and the least frequent events were also calculated. These features were extracted from documents that were translated to German and then back to English. Translator attribution accuracy is

\footnote{http://evllabs.com/jgaap/w}
calculated by using a portion of these documents as training data and the rest as testing data.

There were several experiments carried out with JStylo while using various feature combinations. The first high accuracy yielding feature set was the ‘9-Feature Set’ used by Brennan and Greenstadt [22].

The extracted ‘9-Feature Set’ was classified with SMO using a polynomial kernel by running 10-fold cross-validation. The experiments were performed on a combination of data sets using the Google (google) or Bing (bing) translations where the suffixes en, de, and ja correspond to English, German, and Japanese translations, respectively.

The results of these experiments were the attribution of a text as being translated either by google or bing. Experiments incorporated combinations of features from the ‘9-Feature Set’ and the ‘WritePrints Feature Set’, which is a partial set of features used by Li et al. [69].

‘Translation Feature Set’, shown in Table 2.13, was yielding the highest accuracy after many possible permutations of feature selection, therefore it will be the main feature set throughout translation experiments.

The ‘functions words’ feature consisted of the 512 common function words used by Koppel et al. [62]. For feature classes with many features, such as character bigrams, the class used the top 50 extracted features. These features were also classified with SMO using a polynomial kernel by running 10-fold cross-validation resulting with translator classification as google or bing.

One-way translation translator attribution experiments also used the ‘Translation Feature Set’ on French and Dutch translations performed with Google Translate, Language Weaver and Systran. One-way translation translator attribution experiments used the exact same methods as the two-way translation translator attribution
Authorship Attribution Experiments

Authorship attribution experiments followed the same set of experiments as in translator attribution experiments. The highest accuracy yielding feature set was again the ‘Translation Feature Set’.

2.3.5 Results and Evaluation

Translator Attribution Results

The results support Hedegaard and Simonsen’s [57] suggestion of combining features to increase attribution accuracy. Using a single feature at a time had less successful classification accuracy. JStylo experiments using the ‘Translation Feature Set’ had on average 7% better correct classification than experiments using the ‘9-Feature Set’. The results of the JStylo experiments using the ‘Translation Feature Set’ are as shown in Table 2.14.

The translator attribution results showed higher accuracy for Japanese transla-
Table 2.14: ‘Translation Feature Set’ Translator Attribution

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Correct Attribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>en_de</td>
<td>90.87%</td>
</tr>
<tr>
<td>en_ja</td>
<td>98.80%</td>
</tr>
<tr>
<td>en_ja_de</td>
<td>98.81%</td>
</tr>
<tr>
<td>en_de &amp; en_ja</td>
<td>90.44%</td>
</tr>
<tr>
<td>en_de &amp; en_ja &amp; en_ja_de</td>
<td>91.13%</td>
</tr>
</tbody>
</table>

Translator Attribution Results in One-way Translations

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Correct Attribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>french_translators</td>
<td>92.75%</td>
</tr>
<tr>
<td>dutch_translators</td>
<td>94.44%</td>
</tr>
</tbody>
</table>

Table 2.15: ‘Translation Feature Set’ Translator Attribution on One-way Translations

‘Translation Feature Set’ led to the highest accuracy rate in attributing Google Translate, Language Weaver and Systran, which are as shown in Table 2.15. All other possible feature sets that are available in JStylo led to lower accuracy rates than the ‘Translation Feature Set’.
Authorship Attribution Results

Using a single feature at a time resulted in a correct classification rate close to the random chance rate of 7.69%. JStylo experiments using the ‘9-Feature Set’ had on average a 16% less correct classification rate than experiments using the ‘Translation Feature Set’ in Table 2.13, as shown in Table 2.16. The original writing samples were classified with 97.62% accuracy, labeled as original_text in Table 2.16.

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Correct Attribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>en_de_bing</td>
<td>96.83%</td>
</tr>
<tr>
<td>en_de_google</td>
<td>97.62%</td>
</tr>
<tr>
<td>en_ja_bing</td>
<td>100.00%</td>
</tr>
<tr>
<td>en_ja_google</td>
<td>89.68%</td>
</tr>
<tr>
<td>en_ja_de_bing</td>
<td>77.78%</td>
</tr>
<tr>
<td>en_ja_de_google</td>
<td>87.30%</td>
</tr>
<tr>
<td>all_bing</td>
<td>91.54%</td>
</tr>
<tr>
<td>all_google</td>
<td>91.53%</td>
</tr>
<tr>
<td>all_translations</td>
<td>91.54%</td>
</tr>
<tr>
<td>original_text</td>
<td>97.62%</td>
</tr>
</tbody>
</table>

Table 2.16: ‘Translation Feature Set’ Authorship Attribution

Combining several features when training a classifier led to a higher accuracy than using a single feature for authorship attribution in translated text as was the case for translator attribution. Hedegaard and Simonsen argue that “[f]or translated texts, a combined method of frequent words and frames can outperform methods based solely on traditional markers, on translated texts.” The results outperform Hedegaard and Simonsen’s results using traditional markers in the ‘Translation Feature Set’ shown in Table 2.13 without using context-related features such as semantic frames. A high attribution accuracy is achieved despite an increased translator effect in the corpus.
which contains documents from consequent translations of different languages. An author can be identified with 91.54% accuracy on average compared to Hedegaard and Simonsen’s average authorship attribution accuracy of 75.27% using their proposed feature set.

Hedegaard and Simonsen also suggest that if semantic markers are not used, authorship attribution may not be possible because of the translator footprint. The data set with the most translation iterations was affected thrice by the translator and had the lowest authorship attribution accuracy, demonstrating the validity of Hedegaard and Simonsen’s argument. A broader survey on translator attribution and authorship attribution in translated text which includes semantic features may be conducted if the accuracy continues to decrease as the number of consequent translations on a single document increases. The results of such a survey will depend on the translator’s ability to preserve semantics.

After discovering the ‘Translation Feature Set’ shown in Table 2.13 that yields the highest accuracy for both translator and authorship attribution, WEKA [54] was utilized to calculate the information gain of the features. The comparison of the effectiveness of these features in translator attribution vs. authorship attribution of translated text is as shown in Figure 2.6.

Translator-dependent vs. preserved stylometric features in translated text can be distinguished from the results shown in Figure 2.6.

The preserved stylometric features in descending effect order are mainly: top letter trigrams, words, top letter bigrams; less effectively: function words, letters, and word lengths. Character-count and characters-per-word had a little effect, while punctuation and special characters had no effect. Translator-dependent features in descending effect order are mainly: words, top letter trigrams, function words, and top letter bigrams; less effectively: letters and word lengths. Character-count, characters-
per-word, punctuation, and special characters had little effect.

The comparison shown in Figure 2.6 demonstrates that ‘function words’ are translator-effect-heavy, but less important for authorship attribution. Hedegaard and Simonsen also argues that the impact of the translator may add sufficient noise to render authorship attribution in translated text very difficult. Consequently, excluding a translator-effect-heavy feature such as ‘function words’ should improve authorship attribution in translated text. To test this claim, an additional experiment which excludes function words from the feature set was necessary. The results of this experiment are as shown in Figure 2.7.

The results shown in Figure 2.7 demonstrate that excluding the translator-effect-heavy feature ‘function words’ does not improve authorship attribution. In fact, there is a noticeable decrease in the correct classification rate when ‘function words’
are excluded, suggesting that translator-effect-heavy features do not interfere and can actually aid in more accurate authorship attribution. However, a deeper survey regarding the effects of such features is necessary to arrive at a clearer conclusion.

**Authorship Attribution Results in One-way Translations**

‘Translation Feature Set’ led to the highest accuracy rate in attributing French and Dutch authors using their texts translated to English, which are as shown in Table 2.17. All other possible feature sets available in JStylo led to lower accuracy rates than the ‘Translation Feature Set’. Authorship attribution accuracy of Dutch authors is significantly lower than authorship attribution accuracy of all other authors. As described in the ‘Corpus selection’ section, the Dutch data set possesses different
qualities than the data sets of the other languages. Firstly, the documents have a varying size between 400 and 600 words, whereas the documents of the other data sets are closer to 500 words, which is the optimum length of a document for authorship attribution purposes. Additionally, the essays from each author are on the same six topics: the TV show ‘Big Brother’, smoking, football, a children’s story, ‘Red Riding Hood’, and a historical tale. As a result, word choices and sentences between the author’s essays are very similar. This topic-dependency makes authorship attribution a more difficult task.

Afroz et al. [8] show that authorship attribution is inhibited if an author is trying to imitate another author. Since all the Dutch authors’ essays are about existing stories or cases, they may have been influenced by a dominating style. Such an effect may cause some degree of imitation and render authorship attribution difficult. Also, character count is one of the features in the ‘Translation Feature Set’ and it depends on the length of the document. Since we are using it in the Dutch data set, it causes a misleading effect because of the varying document sizes in this data set.

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Correct Attribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>french_google</td>
<td>100.00%</td>
</tr>
<tr>
<td>french_languageweaver</td>
<td>100.00%</td>
</tr>
<tr>
<td>french_systran</td>
<td>100.00%</td>
</tr>
<tr>
<td>dutch_google</td>
<td>60.42%</td>
</tr>
<tr>
<td>dutch_languageweaver</td>
<td>70.83%</td>
</tr>
<tr>
<td>dutch_systran</td>
<td>75.00%</td>
</tr>
</tbody>
</table>

Table 2.17: ‘Translation Feature Set’ Authorship Attribution on One-way Translations
2.3.6 Conclusion

Machine translation tools introduce an effect on translated text that allows for identifying the machine translation tool used to translate the text. Authorship attribution of translated text is successful given the existence of a translator effect on the text. Translated texts preserve some of original texts’ stylometric features. The more a text goes through iterations of translations, the less preserved the stylometric features become. Machine translation tool attribution and authorship attribution share similar effective features albeit in differing importance levels. These features need to be present in both translator attribution and authorship attribution since they improve attribution accuracy and result in decreased accuracy when removed even though a certain feature may be more effective for either translator or authorship attribution.
2.4 Author Identification in Source Code

This work was completed by Aylin Caliskan-Islam with support from Richard Harang, Andrew Liu, Arvind Narayanan, Clare Voss, and Fabian Yamaguchi. [29].

2.4.1 Introduction

Can the authors of source code be identified automatically through features of their programming style? Do they leave coding “footprints”? Holding important implications for protecting intellectual property as well as for identifying malware authors and tracking how malware spreads and evolves, this question spurred a cross-cutting project involving natural language processing and machine learning. Code stylometry requires features unique to coding and to the programming language. Source code has different properties than common writing, such as the lineage, keywords, comments, the way functions and variables are created, and the grammar of the program. These properties can be used to create a numeric representation of a programmer’s coding style.

Source code authorship attribution has strong privacy and security implications. Contributors to open-source projects may hide their identity whether they are Bitcoin’s creator or just a programmer who does not want her employer to know about her side activities. They may live in a regime that prohibits certain types of software, such as censorship circumvention tools. For example, an Iranian programmer was sentenced to death in 2012 for developing photo sharing software that was used on pornographic websites [117].

On the other hand, source code authorship attribution may be helpful in a forensic context, such as detection of ghostwriting, a form of plagiarism, and investigation of copyright disputes. Malware authors, who leave source code in a breached system, can also be de-anonymized.
Source code authorship attribution has been studied previously. This work represents a qualitative advance over the state-of-the-art by showing that Abstract Syntax Trees (ASTs) carry authorial ‘fingerprints.’ The highest accuracy achieved in the literature is 97%, but this is achieved on a set of only 30 programmers and furthermore relies on using programmer comments and larger amounts of training data [48; 46]. This work matches this accuracy on small programmer sets without this limitation. The largest scale experiments in the literature use 46 programmers and achieve 67.2% accuracy [39]. This work handles orders of magnitude more programmers (1,600) while using less training data with 93.61% accuracy. Furthermore, the coding style features are not trivial to obfuscate. The accuracy remains high while using commercial obfuscators to anonymize source code. While abstract syntax trees can be obfuscated to an extent, doing so incurs significant overhead and maintenance costs.

**Contributions.** First, syntactic features are used to represent coding style. Extracting such features requires parsing of incomplete source code using a fuzzy parser to generate an abstract syntax tree. These features add a component to code stylometry that has so far remained almost completely unexplored. These features are more fundamental and harder to obfuscate. The complete feature set consists of a comprehensive set of around 120,000 layout-based, lexical, and syntactic features. With this complete feature set, a dramatic increase is achieved in accuracy compared to previous work. Second, the method scales to 1,600 programmers without losing much accuracy. Third, this method is not specific to C or C++, and can be applied to any programming language.

C++ source code of thousands of contestants were collected from the annual international competition “Google Code Jam”. A bagging (portmanteau of “bootstrap aggregating”) classifier - random forest was used to attribute programmers to source code. The classifiers reach 98% accuracy in a 250-class closed world task, 94% ac-
accuracy in a 1,600-class closed world task, 100% accuracy on average in a two-class task. Finally, an analysis of various attributes of programmers, types of programming tasks, and types of features that appear to influence the success of attribution is presented. The most important 928 features out of 120,000 are identified and 44% of them are syntactic, 1% are layout-based, and the rest of the features are lexical. 8 training files with an average of 70 lines of code is sufficient for training when using the lexical, layout, and syntactic features. Programmers with a greater skill set are more easily identifiable compared to less advanced programmers and a programmer’s coding style is more distinctive in implementations of difficult tasks as opposed to easier tasks.

2.4.2 Problem Statement

We consider an analyst interested in determining the programmer of an anonymous fragment of source code purely based on its coding style. To do so, the analyst only has access to source code samples with labels of their programmers from a set of candidate programmers, as well as from zero or more unrelated programmers. The analyst determines the programmer of an anonymous fragment of source code by converting each labeled sample into a numerical feature vector, in order to train a machine learning classifier, that can subsequently be used to determine the code’s programmer. This abstract problem formulation captures the following five settings and corresponding applications (see Table 2.18).

**Programmer De-anonymization.** In this scenario, the analyst is interested in determining the identity of an anonymous programmer. For example, if she has a set of programmers who she suspects might be Bitcoin’s creator, Satoshi, and samples of source code from each of these programmers, she could use the initial versions of Bitcoin’s source code to try to determine Satoshi’s identity. Of course, this assumes that
Satoshi did not make any attempts to obfuscate his or her coding style. Given a set of probable programmers, this is considered a closed-world machine learning task with multiple classes where anonymous source code is attributed to a programmer. This is a threat to privacy for open source contributors who wish to remain anonymous.

**Software Forensics.** In software forensics, the analyst assembles a set of candidate programmers based on previously collected malware samples or online code repositories. Unfortunately, she cannot be sure that the anonymous programmer is one of the candidates, making this an open world classification problem as the test sample might not belong to any known category.

**Ghostwriting Detection.** Ghostwriting detection is related to but different from traditional plagiarism detection. The analyst has a suspicious piece of code and one or more candidate pieces of code that the suspicious code may have been plagiarized from. This is a well-studied problem, typically solved using code similarity metrics, as implemented by widely used tools such as MOSS [12], JPlag [95], and Sherlock [93].

For example, a professor may want to determine whether a student’s programming assignment has been written by a student who has previously taken the class. Unfortunately, even though submissions of the previous year are available, the assignments may have changed considerably, rendering code-similarity based methods ineffective. Luckily, stylometry can be applied in this setting—the professor finds the most stylistically similar piece of code from the previous year’s corpus and brings both students in for gentle questioning. Given the limited set of students, this can be considered a closed-world machine learning problem.

**Copyright Investigation.** Theft of code often leads to copyright disputes. Informal arrangements of hired programming labor are very common, and in the absence
of a written contract, someone might claim a piece of code was her own after it was
developed for hire and delivered. A dispute between two parties is thus a two-class
classification problem; where labeled code from both programmers is available to the
forensic expert.

**Authorship Verification.** Finally, the analyst might suspect that a piece of code
was not written by the claimed programmer, but has no leads on who the actual
programmer might be. This is the authorship verification problem. This problem
can be modeled with the textbook approach as a two-class problem where positive
examples come from previous works of the claimed programmer and negative exam-
pies come from randomly selected unrelated programmers. Alternatively, anomaly
detection could be employed in this setting, e.g., using a one-class support vector
machine [109].

As an example, a recent investigation conducted by Verizon on a US company’s
anomalous virtual private network traffic, revealed an employee who was outsourcing
her work to programmers in China. In such cases, training a classifier on employee’s
original code and that of random programmers, and subsequently testing pieces of
recent code, could demonstrate if the employee was the actual programmer.

In each of these applications, the adversary may try to actively modify the pro-
gram’s coding style. In the software forensics application, the adversary tries to
modify code written by her to hide her style. In the copyright and authorship veri-
ification applications, the adversary modifies code written by another programmer to
match his own style. Finally, in the ghostwriting application, two of the parties may
collaborate to modify the style of code written by one to match the other’s style.

---

### Table 2.18: Overview of Applications for Code Stylometry

<table>
<thead>
<tr>
<th>Application</th>
<th>Learner</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>De-anonymization</td>
<td>Multiclass</td>
<td>Closed world</td>
</tr>
<tr>
<td>Software forensics</td>
<td>Multiclass</td>
<td>Open world</td>
</tr>
<tr>
<td>Plagiarism detection</td>
<td>Multiclass</td>
<td>Closed world</td>
</tr>
<tr>
<td>Copyright investigation</td>
<td>Two-class</td>
<td></td>
</tr>
<tr>
<td>Authorship verification</td>
<td>Two-class</td>
<td>One-class also possible</td>
</tr>
</tbody>
</table>

#### 2.4.3 De-anonymizing Programmers

De-anonymizing programmers is possible through creating a machine learning classifier to automatically identify the most likely author of an anonymous source code fragment. The success of machine learning methods in de-anonymizing programmers depend on how accurately the feature set represents the properties of coding style. A fuzzy AST parser, explained in section 2.4.3, parses source code to generate ASTs that reflect programming language use. Section 2.4.3 details the types of extracted features. Finally, the extracted features are used to train a random forest classifier for de-anonymizing the programmers of anonymous source code files.

**Fuzzy Abstract Syntax Trees**

To date, methods for source code authorship attribution focus mostly on sequential feature representations of code such as byte-level and feature level n-grams [47][24]. While these models are well suited to capture naming conventions and preference of keywords, they are entirely language agnostic and thus cannot model author characteristics that become apparent only in the composition of language constructs. For example, an author’s tendency to create deeply nested code, unusually long functions or long chains of assignments cannot be modeled using n-grams alone.

Addressing these limitations requires source code to be parsed. Unfortunately,
int foo ()
{
  if ((x < 0) || x > MAX)
    return -1;

  int ret = bar(x);
  if (ret != 0)
    return -1;
  else
    return 1;
}

Figure 2.8: Sample Code Listing

Figure 2.9: Corresponding Abstract Syntax Tree

parsing C/C++ code using traditional compiler front-ends is only possible for complete code, i.e., source code where all identifiers can be resolved. This severely limits their applicability in the setting of authorship attribution as it prohibits analysis of lone functions or code fragments, as is possible with simple n-gram models.

As a compromise, source code preprocessing employs the fuzzy parser Joern that has been designed specifically with incomplete code in mind [119]. Where possible,
the parser produces *abstract syntax trees* for code fragments while ignoring fragments that cannot be parsed without further information. The produced syntax trees form the basis of the feature extraction procedure.

Consider the function `foo` in Figure 2.8 and a simplified version of the function’s corresponding abstract syntax tree in Figure 2.9. The function contains a number of common language constructs found in many programming languages, such as if-statements (line 3 and 7), return-statements (line 4, 8 and 10), and function call expressions (line 6). For each of these constructs, the abstract syntax tree contains a corresponding node. While the leaves of the tree make classical syntactic features such as keywords, identifiers and operators accessible, inner nodes listed in Table 2.19 represent operations showing how these basic elements are combined to form expressions and statements. In effect, the nesting of language constructs can also be analyzed to obtain a feature set representing the code’s structure.

**Feature Extraction**

Analyzing coding style using machine learning approaches is not possible without a suitable representation of source code that clearly expresses coding style. To address this problem, *Code Stylometry Feature Set* (CSFS), a novel representation of source code is developed specifically for code stylometry. This feature set combines three types of features, namely *lexical features, layout features, and syntactic features*. Lexical and layout features are obtained from source code while the syntactic features can only be obtained from ASTs.

**Lexical and Layout Features**

Numerical features that express preferences for certain identifiers and keywords, as well as some statistics on the use of functions or the nesting depth are extracted from
Table 2.19: Abstract Syntax Tree Node Types

<table>
<thead>
<tr>
<th>AdditiveExpression</th>
<th>AndExpression</th>
<th>Argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArgumentList</td>
<td>ArrayIndexing</td>
<td>AssignmentExpr</td>
</tr>
<tr>
<td>BitAndExpression</td>
<td>BlockStarter</td>
<td>BreakStatement</td>
</tr>
<tr>
<td>Callee</td>
<td>CallExpression</td>
<td>CastExpression</td>
</tr>
<tr>
<td>CastTarget</td>
<td>CompoundStatement</td>
<td>Condition</td>
</tr>
<tr>
<td>ConditionalExpression</td>
<td>ContinueStatement</td>
<td>DoStatement</td>
</tr>
<tr>
<td>ElseStatement</td>
<td>EqualityExpression</td>
<td>ExclusiveOrExpression</td>
</tr>
<tr>
<td>Expression</td>
<td>ExpressionStatement</td>
<td>ForInit</td>
</tr>
<tr>
<td>ForStatement</td>
<td>FunctionDef</td>
<td>GotoStatement</td>
</tr>
<tr>
<td>Identifier</td>
<td>IdentifierDecl</td>
<td>IdentifierDeclStatement</td>
</tr>
<tr>
<td>IdentifierDeclType</td>
<td>IfStatement</td>
<td>IncDec</td>
</tr>
<tr>
<td>IncDecOp</td>
<td>InclusiveOrExpression</td>
<td>InitializerList</td>
</tr>
<tr>
<td>Label</td>
<td>MemberAccess</td>
<td>MultiplicativeExpression</td>
</tr>
<tr>
<td>OrExpression</td>
<td>Parameter</td>
<td>ParameterList</td>
</tr>
<tr>
<td>ParameterType</td>
<td>PrimaryExpression</td>
<td>PtrMemberAccess</td>
</tr>
<tr>
<td>RelationalExpression</td>
<td>ReturnStatement</td>
<td>ReturnType</td>
</tr>
<tr>
<td>ShiftExpression</td>
<td>Sizeof</td>
<td>SizeofExpr</td>
</tr>
<tr>
<td>SizeofOperand</td>
<td>Statement</td>
<td>SwitchStatement</td>
</tr>
<tr>
<td>UnaryExpression</td>
<td>UnaryOp</td>
<td>UnaryOperator</td>
</tr>
<tr>
<td>WhileStatement</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

the source code. Lexical and layout features can be calculated from the source code, without having access to a parser, with basic knowledge of the programming language in use. For example, the number of functions per source line shows the programmer’s preference of longer over shorter functions. Table 2.20 gives an overview of lexical features.

In addition, layout features that represent code-indentation are extracted. For example, layout features determine whether the majority of indented lines begin with whitespace or tabulator characters, and the ratio of whitespace to the file size. Table 2.21 gives a detailed description of these features.
<table>
<thead>
<tr>
<th>Feature</th>
<th>Definition</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>WordUnigramTF</td>
<td>Term frequency of word unigrams in source code</td>
<td></td>
</tr>
<tr>
<td>(\ln(\text{num}\text{keyword}/\text{length}))</td>
<td>Log of the number of occurrences of \text{keyword} divided by file length in characters, where \text{keyword} is one of do, else-if, if, else, switch, for or while</td>
<td>7</td>
</tr>
<tr>
<td>(\ln(\text{num}\text{Ternary}/\text{length}))</td>
<td>Log of the number of ternary operators divided by file length in characters</td>
<td>1</td>
</tr>
<tr>
<td>(\ln(\text{num}\text{Tokens}/\text{length}))</td>
<td>Log of the number of word tokens divided by file length in characters</td>
<td>1</td>
</tr>
<tr>
<td>(\ln(\text{num}\text{Comments}/\text{length}))</td>
<td>Log of the number of comments divided by file length in characters</td>
<td>1</td>
</tr>
<tr>
<td>(\ln(\text{num}\text{Literals}/\text{length}))</td>
<td>Log of the number of string, character, and numeric literals divided by file length in characters</td>
<td>1</td>
</tr>
<tr>
<td>(\ln(\text{num}\text{Keywords}/\text{length}))</td>
<td>Log of the number of unique keywords used divided by file length in characters</td>
<td>1</td>
</tr>
<tr>
<td>(\ln(\text{num}\text{Functions}/\text{length}))</td>
<td>Log of the number of functions divided by file length in characters</td>
<td>1</td>
</tr>
<tr>
<td>(\ln(\text{num}\text{Macros}/\text{length}))</td>
<td>Log of the number of preprocessor directives divided by file length in characters</td>
<td>1</td>
</tr>
<tr>
<td>nestingDepth</td>
<td>Highest degree to which control statements and loops are nested within each other</td>
<td>1</td>
</tr>
<tr>
<td>branchingFactor</td>
<td>Branching factor of the tree formed by converting code blocks of files into nodes</td>
<td>1</td>
</tr>
<tr>
<td>avgParams</td>
<td>The average number of parameters among all functions</td>
<td>1</td>
</tr>
<tr>
<td>stdDevNumParams</td>
<td>The standard deviation of the number of parameters among all functions</td>
<td>1</td>
</tr>
<tr>
<td>avgLineLength</td>
<td>The average length of each line</td>
<td>1</td>
</tr>
<tr>
<td>stdDevLineLength</td>
<td>The standard deviation of the character lengths of each line</td>
<td>1</td>
</tr>
</tbody>
</table>

*About 55,000 for 250 authors with 9 files.

Table 2.20: Lexical Features

Syntactic Features

The syntactic feature set describes the properties of the language dependent abstract syntax tree, and keywords listed in Table 2.23. Calculating these features requires access to an abstract syntax tree. All of these features are invariant to changes in source code layout, as well as comments.

Table 2.22 gives an overview of the syntactic features. These features are generated by preprocessing all C++ source files in the data set to produce their abstract syntax...
trees. An abstract syntax tree is created for each function in the code. There are 58 node types in the abstract syntax tree, listed in Table 2.19, produced by Joern [120].

The AST node bigrams are the most discriminating features of all. AST node bigrams are two AST nodes that are connected to each other. In most cases, when used alone, they provide similar classification results to using the entire feature set.

The term frequency (TF) is the raw frequency of a node found in the abstract syntax tree of each source code file. The term frequency inverse document frequency (TFIDF) of nodes is calculated by multiplying the term frequency of a node by inverse document frequency. The goal in using the inverse document frequency is normalizing the term frequency by the number of authors actually using that particular type of node. The inverse document frequency is calculated by dividing the number of authors in the data set by the number of authors that use that particular node. Consequently, the rarity of a node is captured and the feature is weighted according to its rarity.

The maximum depth of an abstract syntax tree reflects the deepest level a programmer nests a node in the solution. The average depth of the AST nodes shows

<table>
<thead>
<tr>
<th>Feature</th>
<th>Definition</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>ln(numTabs/length)</td>
<td>Log of the number of tab characters divided by file length in characters</td>
<td>1</td>
</tr>
<tr>
<td>ln(numSpaces/length)</td>
<td>Log of the number of space characters divided by file length in characters</td>
<td>1</td>
</tr>
<tr>
<td>ln(numEmptyLines/length)</td>
<td>Log of the number of empty lines divided by file length in characters, excluding leading and trailing lines between lines of text</td>
<td>1</td>
</tr>
<tr>
<td>whiteSpaceRatio</td>
<td>The ratio between the number of whitespace characters (spaces, tabs, and newlines) and non-whitespace characters</td>
<td>1</td>
</tr>
<tr>
<td>newLineBefore OpenBrace</td>
<td>A boolean representing whether the majority of code-block braces are preceded by a newline character</td>
<td>1</td>
</tr>
<tr>
<td>tabsLeadLines</td>
<td>A boolean representing whether the majority of indented lines begin with spaces or tabs</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 2.21: Layout Features
<table>
<thead>
<tr>
<th>Feature</th>
<th>Definition</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaxDepthASTNode</td>
<td>Maximum depth of an AST node</td>
<td>1</td>
</tr>
<tr>
<td>ASTNodeBigramsTF</td>
<td>Term frequency AST node bigrams</td>
<td>dynamic*</td>
</tr>
<tr>
<td>ASTNodeTypesTF</td>
<td>Term frequency of 58 possible AST node type excluding leaves</td>
<td>58</td>
</tr>
<tr>
<td>ASTNodeTypesTFIDF</td>
<td>Term frequency inverse document frequency of 58 possible AST node type</td>
<td>58</td>
</tr>
<tr>
<td>ASTNodeTypeAvgDep</td>
<td>Average depth of 58 possible AST node types excluding leaves</td>
<td>58</td>
</tr>
<tr>
<td>cppkeywords</td>
<td>Term frequency of 84 C++ keywords</td>
<td>84</td>
</tr>
<tr>
<td>CodeInASTLeavesTF</td>
<td>Term frequency of code unigrams in AST leaves</td>
<td>dynamic**</td>
</tr>
<tr>
<td>CodeInASTLeavesTFIDF</td>
<td>Term frequency inverse document frequency of code unigrams in AST leaves</td>
<td>dynamic**</td>
</tr>
<tr>
<td>CodeInASTLeavesAvgDep</td>
<td>Average depth of code unigrams in AST leaves</td>
<td>dynamic**</td>
</tr>
</tbody>
</table>

*About 45,000 for 250 authors with 9 files.
**About 7,000 for 250 authors with 9 files.
**About 4,000 for 150 authors with 6 files.
**About 2,000 for 25 authors with 9 files.

Table 2.22: Syntactic Features

how nested or deep a programmer tends to use particular structural pieces. And lastly, term frequency of each C++ keyword is calculated. Each of these features is written to a feature vector to represent the solution file of a specific author and these vectors are later used in training and testing by machine learning classifiers.

2.4.4 Classification

Using the feature set presented in the previous section, source code can be expressed as numerical vectors, making them accessible to machine learning algorithms. After feature extraction, feature selection is performed to train a random forest classifier capable of identifying the most likely author of a code fragment.
<table>
<thead>
<tr>
<th>alignas</th>
<th>alignof</th>
<th>and</th>
<th>and_eq</th>
<th>asm</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto</td>
<td>bitand</td>
<td>bitor</td>
<td>bool</td>
<td>break</td>
</tr>
<tr>
<td>case</td>
<td>catch</td>
<td>char</td>
<td>char16_t</td>
<td>char32_t</td>
</tr>
<tr>
<td>class</td>
<td>compl</td>
<td>const</td>
<td>constexpr</td>
<td>const_cast</td>
</tr>
<tr>
<td>continue</td>
<td>decltype</td>
<td>default</td>
<td>delete</td>
<td>do</td>
</tr>
<tr>
<td>double</td>
<td>dynamic_cast</td>
<td>else</td>
<td>enum</td>
<td>explicit</td>
</tr>
<tr>
<td>export</td>
<td>extern</td>
<td>false</td>
<td>float</td>
<td>for</td>
</tr>
<tr>
<td>friend</td>
<td>goto</td>
<td>if</td>
<td>inline</td>
<td>int</td>
</tr>
<tr>
<td>long</td>
<td>mutable</td>
<td>namespace</td>
<td>new</td>
<td>noexcept</td>
</tr>
<tr>
<td>not</td>
<td>not_eq</td>
<td>nullptr</td>
<td>operator</td>
<td>or</td>
</tr>
<tr>
<td>or_eq</td>
<td>private</td>
<td>protected</td>
<td>public</td>
<td>register</td>
</tr>
<tr>
<td>reinterpret_cast</td>
<td>return</td>
<td>short</td>
<td>signed</td>
<td>sizeof</td>
</tr>
<tr>
<td>static</td>
<td>static_assert</td>
<td>static_cast</td>
<td>struct</td>
<td>switch</td>
</tr>
<tr>
<td>template</td>
<td>this</td>
<td>thread_local</td>
<td>throw</td>
<td>true</td>
</tr>
<tr>
<td>try</td>
<td>typedef</td>
<td>typeid</td>
<td>typename</td>
<td>union</td>
</tr>
<tr>
<td>unsigned</td>
<td>using</td>
<td>virtual</td>
<td>void</td>
<td>volatile</td>
</tr>
<tr>
<td>wchar_t</td>
<td>while</td>
<td>xor</td>
<td>xor_eq</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.23: C++ Keywords

**Feature Selection**

Due to the heavy use of unigram term frequency and TF/IDF measures, and the diversity of individual terms in the code, the resulting feature vectors are extremely large and sparse, consisting of tens of thousands of features for hundreds of classes. The dynamic *Code stylometry feature set*, for example, produced close to 120,000 features for 250 authors with 9 solution files each.

In many cases, such feature vectors can lead to overfitting (where a rare term, by chance, uniquely identifies a particular author). Extremely sparse feature vectors can also damage the accuracy of random forest classifiers, as the sparsity may result in large numbers of zero-valued features being selected during the random subsampling of the features to select a best split. This reduces the number of ‘useful’ splits that can be obtained at any given node, leading to poorer fits and larger trees. Large, sparse
feature vectors can also lead to slowdowns in model fitting and evaluation, and are often more difficult to interpret. By selecting a smaller number of more informative features, the sparsity in the feature vector can be greatly reduced, thus allowing the classifier to both produce more accurate results and fit the data faster.

Feature selection step is carried out using WEKA’s information gain \[^{[97]}\] criterion, which evaluates the difference between the entropy of the distribution of classes and the entropy of the conditional distribution of classes given a particular feature:

\[
IG(A, M_i) = H(A) - H(A|M_i)
\]

where \( A \) is the class corresponding to an author, \( H \) is Shannon entropy, and \( M_i \) is the \( i^{th} \) feature of the data set. Intuitively, the information gain can be thought of as measuring the amount of information that the observation of the value of feature \( i \) gives about the class label associated with the example.

To reduce the total size and sparsity of the feature vector, features that individually had non-zero information gain were used. These features will be referred to as IG-CSFS throughout the rest of the document. Note that, as \( H(A|M_i) \leq H(A) \), information gain is always non-negative. While the use of information gain on a variable-per-variable basis implicitly assumes independence between the features with respect to their impact on the class label, this conservative approach to feature selection means that the features in use have demonstrable value in classification.

To validate this approach to feature selection, feature selection was applied to two distinct sets of source code files. Sets of features with non-zero information gain were nearly identical between the two sets, and the ranking of features was substantially similar between the two. This suggests that the application of information gain to feature selection is producing a robust and consistent set of features (see Section \[^{[3.7]}\] for further discussion). All the results are calculated by using CSFS and IG-CSFS.
Using IG-CSFS on all experiments demonstrates how these features generalize to different data sets that are larger in magnitude. One other advantage of IG-CSFS is that it consists of a few hundred features that result in non-sparse feature vectors. Such a compact representation of coding style makes de-anonymizing thousands of programmers possible in minutes.

**Random Forest Classification**

Random forest ensemble classifier [20] is used as the main classifier for authorship attribution. Random forests are inherently multi-class classifiers and they do not assume any linear separability in data. They learn nonlinear boundaries. Random forests are ensemble learners built from collections of decision trees, each of which is grown by randomly sampling $N$ training samples with replacement, where $N$ is the number of instances in the data set. To reduce correlation between trees, features are also subsampled; commonly $(\log M) + 1$ features are selected at random (without replacement) out of $M$, and the best split on these $(\log M) + 1$ features is used to split the tree nodes. The number of selected features represents one of the few tuning parameters in random forests: increasing the number of features increases the correlation between trees in the forest which can harm the accuracy of the overall ensemble, however increasing the number of features that can be chosen at each split increases the classification accuracy of each individual tree making them stronger classifiers with low error rates. The optimal range of number of features can be found by using the out of bag (oob) error estimate, or the error estimate derived from those samples not selected for training on a given tree.

During classification, each test example is classified via each of the trained decision trees by following the binary decisions made at each node until a leaf is reached, and the results are then aggregated. The most populous class can be selected as the output
of the forest for simple classification, or classifications can be ranked according to the number of trees that ‘voted’ for a label when performing relaxed attribution (see Section 2.4.5).

Random forests trained on 300 trees, which empirically provided the best trade-off between accuracy and processing time. Examination of numerous oob values across multiple fits suggested that \((\log M) + 1\) random features (where \(M\) denotes the total number of features) at each split of the decision trees was in fact optimal in all of the experiments (listed in Section 3.7), and was used throughout. Node splits were selected based on the information gain criteria, and all trees were grown to the largest extent possible, without pruning.

The data was analyzed via \(k\)-fold cross-validation, where the data was split into training and test sets stratified by author (ensuring that the number of code samples per author in the training and test sets was identical across authors). \(k\) varies according to data sets and is equal to the number of instances present from each author. The cross-validation procedure was repeated 10 times, each with a different random seed. The results section lists average results across all iterations, ensuring that they are not biased by improbably easy or difficult to classify subsets.

### 2.4.5 Evaluation

The evaluation section presents the results to the possible scenarios formulated in the problem statement to evaluate the method. The corpus section gives an overview of the collected data. Section 2.4.5 shows the main results to programmer de-anonymization and how it scales to 1,600 programmers, which is an immediate privacy concern for open source contributors that prefer to remain anonymous. Then, an analysis of training data requirements and efficacy of types of features is performed. The obfuscation section discusses a possible countermeasure to programmer
de-anonymization. Possible machine learning formulations along with the verification section that extends the approach to an open world problem is discussed. The evaluation ends with generalizing the method to other programming languages and providing software engineering insights.

Corpus

One concern in source code authorship attribution is that we are actually identifying differences in coding style, rather than merely differences in functionality. Consider the case where Alice and Bob collaborate on an open source project. Bob writes user interface code whereas Alice works on the network interface and backend analytics. If we used a data set derived from their project, we might differentiate differences between frontend and backend code rather than differences in style.

In order to minimize these effects, the evaluation is performed on the source code of solutions to programming tasks from the international programming competition Google Code Jam (GCJ), made public in 2008 [1]. The competition consists of algorithmic problems that need to be solved in a programming language of choice. In particular, this means that all programmers solve the same problems, and hence implement similar functionality, a property of the data set crucial for code stylometry analysis. The release of these results presented opportunity to build a source code corpus with thousands of programmers with ground truth information on authorship.

The data set contains solutions by professional programmers, students, academics, and hobbyists from 166 countries. The majority of the contestants were from India, United States, China, Russia, Japan, Canada, Brasil, South Korea, France, Egypt, and Poland. Participation statistics are similar over the years. Moreover, it contains problems of different difficulty, as the contest takes place in several rounds. This makes it possible to assess whether coding style is related to programmer experience
and problem difficulty.

Programmers need to pass the qualification round within a 27 hour frame to become contestants and advance to the online rounds. 3,000 contestants from the first round that have the highest scores advance to the second round. The top-scoring 500 contestants in the second round advance to the third round. 25 of the top-scoring contestants in the third round advance to the onsite final round. As the round number increases, the set of problems become more difficult. For example, 26,470 contestants were able to pass the qualification round. 15,563 of these completed round-1 and 3,000 contestants with the top scores advanced to round-2. Only 2,599 contestants out of this set of skilled 3,000 contestants were able to complete round-2. 500 contestants with the highest scores from round-2 advanced to round-3 and only 393 of these highly skilled programmers were able to complete round-3.

The most commonly used programming language was C++, followed by Java, and Python. C++ and C data was collected because of their popularity in the competition and having a parser for C/C++ readily available [119]. Some preliminary experimentation was conducted on Python as well.

A validation data set was created from 2012’s GCJ competition. Some problems had two stages, where the second stage involved answering the same problem in a limited amount of time and for a larger input. The solution to the large input is essentially a solution for the small input but not vice versa. Therefore, collecting both of these solutions could result in duplicate and identical source code. In order to avoid multiple entries, only small input versions’ solutions were used in the data set.

The programmers had up to 19 solution files in these data sets. Solution files have an average of 70 lines of code per programmer.

To create the experimental data sets that are discussed in further detail in the
results section;

(i) The corpus was first partitioned by year of competition. The “main” data set includes files drawn from 2014 (250 programmers). The “validation” data set files come from 2012, and the “multi-year” data set files come from years 2008 through 2014 (1,600 programmers).

(ii) Within each year, the corpus files are ordered by the round in which they were written, and by the problem within a round, as all competitors proceed through the same sequence of rounds in that year. As a result, stratified cross validation is performed on each program file by the year it was written, by the round in which the program was written, by the problems solved in the round, and by the author’s highest round completed in that year.

**Programmer De-anonymization**

This section presents the main experiment—de-anonymizing 250 programmers in the difficult scenario where all programmers solved the same set of problems. The biggest data set formed from 2014’s Google Code Jam Competition with 9 solution files to the same problem had 250 programmers. These were the easiest set of 9 problems, making the classification more challenging (see Section 2.4.5). *Code Stylometry Feature Set* reached 91.78% accuracy in classifying 250 programmers. After applying information gain and using the features that had positive information gain, the accuracy was 95.08%.

Another data set had 250 programmers from different years and randomly selected 9 solution files for each one of them. The information gain features obtained from 2014’s data set were used to see how well they generalize. IG-CSFS reached 98.04% accuracy in classifying 250 programmers. This is 3% higher than the controlled large data set’s results. The accuracy might be increasing because of using a mixed set
of Google Code Jam problems, which potentially contains the possible solutions’ properties along with programmers’ coding style and makes the code more distinct.

To evaluate the approach and validate the method and important features, a data set from 2012’s Google Code Jam Competition with 250 programmers who had the solutions to the same set of 9 problems, was created. The feature set consisted of only the features that had positive information gain in 2014’s data set, which was used as the main data set to implement the approach. The classification accuracy was 96.83%, which is higher than the 95.07% accuracy obtained in 2014’s data set.

The high accuracy of validation results in Table 2.24 show that the important features of code stylometry and a stable feature set are identified. This feature set does not necessarily represent the exact features for all possible data sets. For a given data set that has ground truth information on authorship, following the same approach should generate the most important features that represent coding style in that particular data set.

<table>
<thead>
<tr>
<th></th>
<th>A = #programmers, F = max #problems completed</th>
<th>N = #problems included in data set (N ≤ F)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A = 250 from 2014</strong></td>
<td><strong>A = 250 from 2012</strong></td>
<td><strong>A = 250 all years</strong></td>
</tr>
<tr>
<td><strong>F = 9 from 2014</strong></td>
<td><strong>F = 9 from 2014</strong></td>
<td><strong>F ≥ 9 all years</strong></td>
</tr>
<tr>
<td>N = 9</td>
<td>N = 9</td>
<td>N = 9</td>
</tr>
<tr>
<td>Average accuracy after 10 iterations with IG-CSFS features</td>
<td>95.07%</td>
<td>96.83%</td>
</tr>
</tbody>
</table>

Table 2.24: Validation Experiments
Scaling

Collection of a larger data set of 1,600 programmers from various years was crucial to perform large scale experiments. Each of the programmers had 9 source code samples. The large data set was divided to 6 subsets in differing sizes, with 250 programmers, 500 programmers, 750 programmers, 1,250 programmers, 1,500 programmers, and 1,600 programmers. These subsets are useful to understand how well the approach scales. The specific features that had information gain in the main 250 programmer data set were extracted from this large data set. In theory, a classifier needs to use more trees in the random forest as the number of classes increase to decrease variance, but in this experiment, the classifier used fewer trees compared to smaller experiments. A random forest of 300 trees was used to run the experiments in a reasonable amount of time with a reasonable amount of memory. The accuracy did not decrease too much when increasing the number of programmers. This result shows that information gain features are robust against changes in class and are important properties of programmers’ coding styles. The following Figure 2.10 demonstrates how well the method scales. The classifier is able to de-anonymize 1,600 programmers using 32GB memory within one hour. Alternately, using a classifier with 40 trees leads to nearly the same accuracy (within 0.5%) in a few minutes.

Training Data and Features

Different data sets are formed that consisted of different sets of 62 programmers who had F solution files, from 2 up to 14. Each data set has the solutions to the same set of F problems by different sets of programmers. Each data set consists of programmers that were able to solve exactly F problems. Such an experimental setup makes it possible to investigate the effect of programmer skill set on coding style. The size of the data sets are limited to 62, because there are only 62 contestants with 14
files. There are a few contestants with up to 19 files but they cannot be included in the data set since there were not enough programmers to compare them.

The same set of $F$ problems are used to ensure that the coding style of the programmer is being classified and not the properties of possible solutions of the problem itself. The feature set is able to capture personal programming style since all the programmers are coding the same functionality in their own ways.

Stratified $F$-fold cross validation was used by training on everyone’s $(F - 1)$ solutions and testing on the $F^{th}$ problem that did not appear in the training set. As a result, the problems in the test files were encountered for the first time by the classifier.

A random forest with 300 trees and $(\log M) + 1$ features performed $F$-fold stratified cross validation, first with the Code Stylometry Feature Set (CSFS) and then with the CSFS’s features that had information gain.

Figure 2.11 shows the accuracy from 13 different sets of 62 programmers with 2 to 14 solution files, and consequently 1 to 13 training files. The CSFS reaches an optimal training set size at 9 solution files, where the classifier trains on $8 \ (F - 1)$ solutions.
In the data sets we constructed, as the number of files increase and problems from more advanced rounds are included, the average line of code (LOC) per file also increases. The average lines of code per source code in the data set is 70. Increased number of lines of code might have a positive effect on the accuracy but at the same time it reveals programmer’s choice of program length in implementing the same functionality. On the other hand, the average line of code of the 7 easier (76 LOC) or difficult problems (83 LOC) taken from contestants that were able to complete 14 problems, is higher than the average line of code (68) of contestants that were able to solve only 7 problems. This shows that programmers with better skills tend to write longer code to solve Google Code Jam problems. The mainstream idea is that better programmers write shorter and cleaner code which contradicts with line of code statistics in these data sets. Google Code Jam contestants are supposed to optimize their code to process large inputs with faster performance. This implementation strategy might be leading to advanced programmers implementing longer solutions for the sake of optimization.

On the data set with 62 programmers each with 9 solutions, the classification ac-
accuracy is 97.67% with all the features and 99.28% with the information gain features. Excluding all the syntactic features decreases the accuracy to 88.89%. Taking the information gain of all non-syntactic features lead to 88.35% accuracy. Excluding all the non-syntactic features and using only the syntactic features resulted in 96.06% accuracy. Taking the information gain of all the syntactic features lead to 96.96% accuracy. Most of the classification power is preserved with the syntactic features, and using non-syntactic features leads to a significant decline in accuracy.

**Obfuscation**

An off-the-shelf C++ obfuscator called stunnix [5] was used to obfuscate the code of a data set with 9 solution files and 25 programmers. The accuracy with the information gain code stylometry feature set on the obfuscated data set is 97.77%. The accuracy on the same data set when the code is not obfuscated is 98.67%. The obfuscator refactored function and variable names, as well as comments, and stripped all the spaces, preserving the functionality of code without changing the structure of the program. Obfuscating the data produced little detectable change in the performance of the classifier for this sample. The results are summarized in Table 2.25.

A much more sophisticated open source obfuscator called Tigress [2] was used to obfuscate the code of another data set with 9 solution files in C and 20 programmers (see example in Figures 2.12 and 2.13). In particular, Tigress implements function virtualization, an obfuscation technique that turns functions into interpreters and converts the original program into corresponding bytecode. After applying function virtualization, de-anonymizing programmers became less accurate, so it has potential as a countermeasure to programmer de-anonymization. However, this obfuscation comes at a cost. First of all, the obfuscated code is neither readable nor maintainable, and is thus unsuitable for an open source project. Second, the obfuscation adds
significant overhead (9 times slower) to the runtime of the program, which is another disadvantage.

```c
#include<stdio.h>
int main()
{
    int T, test = 1;
    double C, F, X, rate, time;
    scanf("%d", &T);
    while (T--)
    {
        scanf("%lf %lf %lf", &C, &F, &X);
        rate = 2.0;
        time = 0;
        while (X/rate > C/rate + X/(rate + F))
        {
            time += C/rate;
            rate += F;
        }
        time += X/rate;
        printf("Case #%d: %lf\n", test++, time);
    }
    return 0;
}
```

Figure 2.12: A Code Sample $X$

Figure 2.12 shows a source code sample $X$ from the data set that is 21 lines long. After obfuscation with Tigress, sample $X$ became 537 lines long. Figure 2.13 shows the first 14 lines of the obfuscated sample $X$.

The accuracy with the information gain feature set on the obfuscated data set is reduced to 67.22%. When the feature set is limited to AST node bigrams, de-anonymization accuracy drops to 18.89%, which demonstrates the need for all feature types in certain scenarios. The accuracy on the same data set when the code is not obfuscated is 95.91%.
Two-class Classification

Source code author identification could automatically deal with source code copyright disputes without requiring manual analysis by a neutral code investigator. A
<table>
<thead>
<tr>
<th>Obfuscator</th>
<th>Programmers</th>
<th>Language</th>
<th>Results w/o Obfuscation</th>
<th>Results w/ Obfuscation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stunnix</td>
<td>25</td>
<td>C++</td>
<td>96.89%</td>
<td>95.56%</td>
</tr>
<tr>
<td>Stunnix</td>
<td>25</td>
<td>C++</td>
<td>98.67*%</td>
<td>97.77*%</td>
</tr>
<tr>
<td>Tigress</td>
<td>20</td>
<td>C</td>
<td>93.65%</td>
<td>58.33%</td>
</tr>
<tr>
<td>Tigress</td>
<td>20</td>
<td>C</td>
<td>95.91*%</td>
<td>67.22*%</td>
</tr>
</tbody>
</table>

*Information gain features

Table 2.25: Effect of Obfuscation on De-anonymization

copyright dispute on code ownership can be resolved by comparing the styles of both parties claiming to have generated the code. The style of the disputed code can be compared to both parties’ former source code to aid in the investigation. To imitate such a scenario, a data set was formed from 60 different pairs of programmers, each with 9 solution files. A random forest performed 9-fold cross validation to classify two programmers’ source code. The average classification accuracy using CSFS set is 100.00% and accuracy using information gain features is also 100.00%.

Another two-class machine learning task can be formulated for authorship verification. We suspect Mallory of plagiarizing, so we mix in some code of hers with a large sample of other people’s, test, and see if the disputed code gets classified as hers or someone else’s. If it gets classified as hers, then it was with high probability really written by her. If it is classified as someone else’s, it really was someone else’s code. This could be an open world problem and the person that originally wrote the code could be a previously unknown programmer.

This is a two-class problem with classes Mallory and others. A random forest trains on Mallory’s solutions to problems a, b, c, d, e, f, g, h. The random forest also trains on programmer A’s solution to problem a, programmer B’s solution to problem b, programmer C’s solution to problem c, programmer D’s solution to problem d, programmer E’s solution to problem e, programmer F’s solution to problem f, programmer G’s solution to problem g, programmer H’s solution to problem h and puts
them in one class called ABCDEFGH. The random forest classifier with 300 trees trains on classes Mallory and ABCDEFGH. There are 6 test instances from Mallory and 6 test instances from another programmer ZZZZZZ, who is not in the training set.

These experiments have been repeated in the exact same setting with 80 different sets of programmers ABCDEFGH, ZZZZZZ and Mallorys. The average classification accuracy for Mallory using the CSFS set is 100.00%. ZZZZZZ’s test instances are classified as programmer ABCDEFGH 82.04% of the time, and classified as Mallory for the rest of the time while using the CSFS. Depending on the amount of acceptable false positives, the operating point on the ROC curve can be adjusted.

These results are also promising for use in cases where a piece of code is suspected to be plagiarized. Following the same approach, if the classification result of the piece of code is someone other than Mallory, that piece of code was with very high probability not written by Mallory.

**Verification/Open World Problem**

In a real world scenario, we do not know if source code belongs to one of the programmers’ in the suspect set. In such cases, the classifier can classify the anonymous source code, and if the majority number of votes of trees in the random forest is below a certain threshold, the classifier can reject the classification considering the possibility that it might not belong to any one of the classes in the training data. By doing so, the approach scales to an open world scenario, where we a suspect might not have been encountered before. As long as a confidence threshold is determined based on training data [109], the probability that an instance belongs to one of the programmers in the set can be calculated and accordingly the classifier can accept or reject the classification.
270 classifications are performed in a 30-class problem using all the features to determine the confidence threshold based on the training data. The accuracy was 96.67%. There were 9 misclassifications and all of them were classified with less than 15% confidence by the classifier. The class probability or classification confidence \( P(C)_i \) is calculated by taking the percentage of trees \( T \) in the random forest \( f \) that voted for that particular class \( V_i \), which can be seen in equation \( 2.5 \)

\[
P(C)_i = \frac{\sum V_i}{\sum T_f} \quad (2.5)
\]

There was one correct classification made with 13.7% confidence. This suggests that a threshold between 13.7% and 15% confidence level can be used for verification, and the classifications that did not pass the confidence threshold can be manually analyzed or excluded from results.

An aggressive threshold of 15% was picked and to validate the threshold, a random forest classifier trained on the same set of 30 programmers with 270 code samples. The classifier tested 150 different files from the programmers in the training set. There were 6 classifications below the 15% threshold and two of them were misclassified. Another test set consisted of 420 test files from 30 programmers that were not in the training set. All the files from these 30 programmers were attributed to one of the 30 programmers in the training set since this is a closed world classification task, however, the highest confidence level in these classifications was 14.7%. The 15% threshold catches all the instances that do not belong to the programmers in the suspect set, gets rid of 2 misclassifications and 4 correct classifications. Consequently, if a classification is made with a confidence value less than a certain threshold, the classification can be rejected and the test instance can be attributed to an unknown suspect.
Relaxed Classification

The goal here is to determine whether it is possible to reduce the number of suspects using code stylometry. Reducing the set of suspects in challenging cases, such as having too many suspects, would reduce the effort required to manually find the actual programmer of the code.

In this section, a random forest classifies the 250 programmers in the main data set from 2014 using the information gain features. The classification was relaxed to a set of top $R$ suspects instead of exact classification of the programmer. The relaxed factor $R$ varied from 1 to 10. Instead of taking the highest majority vote of the decisions trees in the random forest, the highest $R$ majority vote decisions were taken and the classification result was considered correct if the programmer was in the set of top $R$ highest voted classes. The accuracy does not improve much after the relaxed factor is larger than 5.

![Figure 2.14: Relaxed Classification with 250 Programmers](image)

Generalizing the Method

Features derived from ASTs can represent coding styles in various languages. These features are applicable in cases when lexical and layout features may be less
discriminating due to formatting standards and reliance on whitespace and other ‘lexical’ features as syntax, such as Python’s PEP8 formatting. To show that the method generalizes, source code of 229 Python programmers was collected from GCJ’s 2014 competition. 229 programmers had exactly 9 solutions. Using only the Python equivalents of syntactic features listed in Table 2.22 and 9-fold cross-validation, the average accuracy is 53.91% for top-1 classification, 75.69% for top-5 relaxed attribution. The largest set of programmers to all work on the same set of 9 problems was 23 programmers. The average accuracy in identifying these 23 programmers is 87.93% for top-1 and 99.52% for top-5 relaxed attribution. The same classification tasks using the information gain features are also listed in Table 2.26. The overall accuracy in data sets composed of Python code are lower than C++ data sets. In Python data sets, a parser that was not fuzzy generated the ASTs, which had an effect on the syntactic features. The lack of quantity and specificity of features accounts for the decreased accuracy. The Python data set’s information gain features are significantly fewer in quantity, compared to C++ data set’s information gain features. Information gain only keeps features that have discriminative value all on their own. If two features only provide discriminative value when used together, then information gain will discard them. So if a lot of the features for the Python set are only jointly discriminative (and not individually discriminative), then the information gain criteria may be removing features that in combination could effectively discriminate between authors. This might account for the decrease when using information gain features. Nevertheless, a CSFS equivalent feature set can be generated for other programming languages by implementing the layout and lexical features as well as using a fuzzy parser.
<table>
<thead>
<tr>
<th>Language</th>
<th>Programmers</th>
<th>Classification</th>
<th>IG</th>
<th>Top-5</th>
<th>Top-5 IG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python</td>
<td>23</td>
<td>87.93%</td>
<td>79.71%</td>
<td>99.52%</td>
<td>96.62</td>
</tr>
<tr>
<td>Python</td>
<td>229</td>
<td>53.91%</td>
<td>39.16%</td>
<td>75.69%</td>
<td>55.46</td>
</tr>
</tbody>
</table>

Table 2.26: Generalizing to Other Programming Languages

**Software Engineering Insights**

Is programming style consistent throughout years? There were some contestants that had the same username and country information both in 2012 and 2014. In 2014, someone else might have picked up the same username from the same country and started using it. Assuming that these are the same people, we are going to ignore such a ground truth problem for now and assume that they are the same people.

A data set was created from a set of 25 programmers from 2012 that were also contestants in 2014’s competition. A random forest classifier trained on the 8 files from their submissions in 2012, with 300 trees using CSFS. The test documents consisted of one instance from each one of the contestants from 2014. The correct classification rate of these test instances from 2014 is 96.00%. The accuracy dropped to 92.00% when using only information gain features, which might be due to the aggressive elimination of pairs of features that are jointly discriminative. These 25 programmers’ 9 files from 2014 had a correct classification accuracy of 98.04%. These results indicate that coding style is preserved up to some degree throughout years.

To investigate problem difficulty’s effect on coding style, two data sets were created from 62 programmers that had exactly 14 solution files. Table 2.27 summarizes the following results. A data set with 7 of the easier problems out of 14 resulted in 95.62% accuracy. A data set with 7 of the more difficult problems out of 14 resulted in 99.31% accuracy. This might imply that more difficult coding tasks have a more prevalent reflection of coding style. On the other hand, the data set that had 62
programmers with exactly 7 of the easier problems resulted in 91.24% accuracy, which is a lot lower than the accuracy obtained from the data set whose programmers were able to advance to solve 14 problems. This might indicate that, programmers who are advanced enough to answer 14 problems likely have more unique coding styles compared to contestants that were only able to solve the first 7 problems.

To investigate the possibility that contestants who are able to advance further in the rounds have more unique coding styles, a second round of experiments were performed on comparable data sets. A data set was created from 62 programmers that had 12 solution files. The subset of this data set with 6 of the easier problems out of 12 resulted in 91.39% accuracy. The subset of this data set with 6 of the more difficult problems out of 12 resulted in 94.35% accuracy. These results are higher than the data set whose programmers were only able to solve the easier 6 problems. The data set that had 62 programmers with exactly 6 of the easier problems resulted in 90.05% accuracy.

<table>
<thead>
<tr>
<th>A = #programmers, F = max #problems completed</th>
<th>N = #problems included in data set (N ≤ F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A = 62</td>
<td></td>
</tr>
<tr>
<td>F = 14</td>
<td>F = 7</td>
</tr>
<tr>
<td>N = 7</td>
<td>N = 7</td>
</tr>
<tr>
<td>Average accuracy after 10 iterations while using CSFS</td>
<td></td>
</tr>
<tr>
<td>99.31%</td>
<td>95.62%</td>
</tr>
<tr>
<td>91.24%</td>
<td>94.35%</td>
</tr>
<tr>
<td>91.39%</td>
<td>90.05%</td>
</tr>
<tr>
<td>2 Drop in accuracy due to programmer skill set.</td>
<td></td>
</tr>
<tr>
<td>2 Coding style is more distinct in more difficult tasks.</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.27: Effect of Problem Difficulty on Coding Style
2.4.6 Discussion

**Problem Difficulty.** The experiment with random problems from random authors among seven years most closely resembles a real world scenario. In such an experimental setting, there is a chance that instead of only identifying authors, the classifier is also identifying the properties of a specific problem’s solution, which results in a boost in accuracy.

In contrast, the main experimental setting where all authors have only answered the nine easiest problems is possibly the hardest scenario, since the classifier is training on the same set of eight problems that all the authors have algorithmically solved and tries to identify the authors from the test instances that are all solutions of the 9th problem. On the upside, these test instances help precisely capture the differences between individual coding style that represent the same functionality. The results also reflect that such a scenario is harder since the randomized data set has higher accuracy.

Classifying authors that have implemented the solution to a set of difficult problems is easier than identifying authors with a set of easier problems. This shows that coding style is reflected more through difficult programming tasks. This might indicate that programmers come up with unique solutions and preserve their coding style more when problems get harder. On the other hand, programmers with a better skill set have a prevalent coding style which can be identified more easily compared to contestants who were not able to advance as far in the competition. This might indicate that as programmers become more advanced, they build a stronger coding style compared to novices. There is another possibility that maybe better programmers start out with a more unique coding style.
Effects of Obfuscation. A malware author or plagiarizing programmer might deliberately try to hide his source code by obfuscation. Our experiments indicate that this method is resistant to simple off-the-shelf obfuscators such as stunnix, that make code look cryptic while preserving functionality. The reason for this success is that the changes stunnix makes to the code have no effect on syntactic features, e.g., removal of comments, changing of names, and stripping of whitespace.

In contrast, sophisticated obfuscation techniques such as function virtualization hinder de-anonymization to some degree, however, at the cost of making code unreadable and introducing a significant performance penalty. Unfortunately, unreadability of code is not acceptable for open-source projects, while it is no problem for attackers interested in covering their tracks. Developing methods to automatically remove stylometric information from source code without sacrificing readability is therefore a promising direction for future research.

Limitations. The case where a source file might be written by a different author than the stated contestant is a ground truth problem that we cannot control. Moreover, it is often the case that code fragments are the work of multiple authors. To shed light on the feasibility of classifying such code, analyzing a data set formed of git commits to open source projects is one possible direction. Such an experiment should be possible since Joern, the parser that was utilized throughout the experiments, works on code fragments rather than complete code.

Another fundamental problem for machine learning classifiers are mimicry attacks. For example, the random forest classifier may be evaded by an adversary by adding extra dummy code to a file that closely resembles that of another programmer, albeit without affecting the program’s behavior. This evasion is possible, but trivial to resolve when an analysts verifies the decision.

Finally, verifying authorship information of Google Code Jam contestants is not
possible. In this case, a classify and then verify approach as explained in Stolerman et al.’s work [109] is helpful. Each classification could go through a verification step to eliminate instances where the classifier’s confidence is below a threshold. After the verification step, instances that do not belong to the set of known authors can be separated from the data set to be excluded or to be further manually analyzed.

2.4.7 Related Work

Programmer de-anonymization is inspired by the research done on authorship attribution of unstructured or semi-structured text [81,9]. This section discusses prior work on source code authorship attribution. In general, previous work on programmer de-anonymization (Table 2.28) looks at smaller scale problems, does not use structural features, and achieves lower accuracies.

The highest accuracies in the related work are achieved by Frantzeskou et al. [48,46]. They used 1,500 7-grams to reach 97% accuracy with 30 programmers. They investigated the high-level features that contribute to source code authorship attribution in Java and Common Lisp. They determined the importance of each feature by iteratively excluding one of the features from the feature set. They showed that comments, layout features and naming patterns have a strong influence on the author classification accuracy. They used more training data (172 lines of code on average) than us (70 lines of code). This work replicated their experiments on a 30 programmer subset of the C++ data set, with eleven files containing 70 lines of code on average and no comments. The classifier reaches 76.67% accuracy with 6-grams, and 76.06% accuracy with 7-grams. When a 6 and 7-gram feature set were used on 250 programmers with 9 files, the accuracy decreased to 63.42%. A random forest with 300 tress, incorporating CSFS, reaches 98% accuracy on 250 programmers.

The largest number of programmers studied in the related work was 46 program-
mers with 67.2% accuracy. Ding and Samadzadeh [39] use statistical methods for authorship attribution in Java. They show that among lexical, keyword and layout properties, layout metrics have a more important role than others which is not the case in our analysis.

There are also a number of smaller scale, lower accuracy approaches in the literature [25; 64; 105; 65; 72; 42; 67], shown in Table 2.28, all of which this work significantly outperforms. These approaches use a combination of layout and lexical features.

The only other work to explore structural features is by Pellin [88], who used manually parsed abstract syntax trees with an SVM that has a tree based kernel to classify functions of two programmers. He obtains an average of 73% accuracy in a two class classification task. His approach explained in the white paper can be extended to incorporate CSFS, so it is the closest to this work in the literature. This work demonstrates that it is non-trivial to use ASTs effectively and is the first to use structural features to achieve higher accuracies at larger scales and the first to study how code obfuscation affects code stylometry.

There has also been some code stylometry work that focused on manual analysis and case studies. Spafford and Weeber [107] suggest that use of lexical features such as variable names, formatting and comments, as well as some syntactic features such as usage of keywords, scoping and presence of bugs could aid in source code attribution but they do not present results or a case study experiment with a formal approach. Gray et al. [53] identify three categories in code stylometry: the layout of the code, variable and function naming conventions, types of data structures being used and also the cyclomatic complexity of the code obtained from the control flow graph. They do not mention anything about the syntactic characteristics of code, which could potentially be a great marker of coding style that reveals the usage of programming
language’s grammar. Their case study is based on a manual analysis of three worms, rather than a statistical learning approach. Hayes and Offutt [56] examine coding style in source code by their consistent programmer hypothesis. They focused on lexical and layout features, such as the occurrence of semicolons, operators and constants. Their data set consisted of 20 programmers and the analysis was not automated. They concluded that coding style exists through some of their features and professional programmers have a stronger programming style compared to students. In the results in section 2.4.5, this work also shows that more advanced programmers have a more identifying coding style.

There is also a great deal of research on plagiarism detection which is carried out by identifying the similarities between different programs. For example, there is a widely used tool called Moss that originated from Stanford University for detecting software plagiarism. Moss [12] is able to analyze the similarities of code written by different programmers. Rosenblum et al. [101] present a novel program representation and techniques that automatically detect the stylistic features of binary code.

2.4.8 Conclusion and Future Work

Source code stylometry has direct applications for privacy, security, software forensics, plagiarism, copyright infringement disputes, and authorship verification. Source code stylometry is an immediate concern for programmers who want to contribute code anonymously because de-anonymization is quite possible. This work introduces the first principled use of syntactic features along with lexical and layout features to investigate style in source code. 1,600 programmers can be de-anonymized with 94% accuracy in and 250 programmers with 98% accuracy with eight training files per class. This shows that source code authorship attribution with the Code Stylometry Feature Set scales even better than regular stylometric authorship attribution,
Related Work | # of Programmers | Results
---|---|---
Pellin [88] | 2 | 73%
MacDonell et al. [72] | 7 | 88.00%
Frantzeskou et al. [48] | 8 | 100.0%
Burrows et al. [25] | 10 | 76.78%
Elenbogen and Seliya [42] | 12 | 74.70%
Kothari et al. [64] | 12 | 76%
Lange and Mancoridis [67] | 20 | 75%
Krsul and Spafford [65] | 29 | 73%
Frantzeskou et al. [48] | 30 | 96.9%
Ding and Samadzadeh [39] | 46 | 67.2%
This work | 8 | 100.00%
This work | 35 | 100.00%
This work | 250 | 98.04%
This work | 1,600 | 92.83%

Table 2.28: Comparison to Previous Results

as these methods can only identify individuals in sets of 50 authors with slightly over 90% accuracy [6]. Furthermore, this performance is achieved by training on only 550 lines of code or eight solution files, whereas classical stylometry requires 5,000 words of training data.

Additionally, the results in this work raise a number of questions that motivate future research. First, as malicious code is often only available in binary format, it would be interesting to investigate whether syntactic features can be partially preserved in binaries. This may require the feature set to be improved in order to incorporate information obtained from control flow graphs.

Second, can the classification accuracy be further increased? For example, does using features that have joint information gain alongside features that have information gain by themselves improve performance? Moreover, designing features that capture larger fragments of the abstract syntax tree could provide improvements. These changes (along with adding lexical and layout features) may provide signifi-
cant improvements to the Python results and help generalize the approach further.

Finally, investigating whether code can be automatically normalized to remove stylistic information while preserving functionality and readability will be a first step towards anonymizing code while preserving readability.
3. Modeling and Quantifying Privacy Behavior

This work was completed by Aylin Caliskan-Islam with support from Jonathan Walsh. [29].

Analysis on the amount of private information shared by Twitter users [29] showed that friends who share similar privacy scores appear in clusters. People also tend to mention friends with similar privacy scores in their tweets. This correlation presents a starting point to investigate the causation behind revealing private information in online social networks.

A collection of timelines along with friend lists of 500,000 Twitter users through the Twitter API was used to analyze the users’ privacy revealing habits. Ten privacy categories, based on a general societal consensus on what is private supported by related work, were used to label tweets as private or not. After having Amazon Mechanical Turk workers annotate timelines of 270 users, a privacy score calculator assigned a score from one, being mostly public, to 3, being mostly private to these labeled timelines. A machine learning classifier used this ground truth representation to associate 1,982 Twitter users with a privacy score. The numeric representation of privacy score consisted of privacy related features that can be extracted through natural language processing techniques. One such feature is named entities mentioned in text, the more entities named in a text, the more specific the descriptiveness of that text becomes. Named entity recognition identifies elements in text such as persons, organizations, and locations. Another feature is the distribution of topics a user talks about. Some topics tend to be more personal, such as medical information, emotions, politics, and religious views whereas some topics are public and do not contain as much private data, such as sports, weather, and news. Detecting topics and named
entities alone is not sufficient for understanding the underlying meaning of text, which in aggregate forms privacy behavior. This is where Brown clustering and semantic classification become useful. Brown clustering is a distributional similarity method that groups words appearing in the same context. Semantic classification exposes the intention in sentences.

The privacy score classifier reached 95% correct classification accuracy on the labeled data of 270 users in cross validation by using natural language processing based features. Such a learning based approach covering a wide range of privacy concerns has not been established before. Previous work focused on keyword based detection of limited privacy categories, such as location privacy, medical privacy, or writing under the influence. A learning based privacy score calculator could utilize a user’s timeline to guide her during the selection of privacy settings while tailoring user’s needs to the privacy policy in hand. In future work, this can be extended to automate privacy policy understanding and create a privacy setting management tool. Currently, a Facebook study has been initiated to investigate the influencing factors of private information disclosure. This study particularly aims to answer, do people who tend to reveal private information influence their friends to share more private information and what are the other causal factors behind privacy behavior?

3.1 Introduction

Numerous organizations, from corporations to governments to criminal gangs, are actively engaged in the collection of personal information released on the Internet. Generally, this pervasive collection is performed without the user’s knowledge. Internet users need an increased ability to realize how they are influenced to reveal privacy and the amount of sensitive information they are exposing.
Twitter users with public accounts expose user information through tweets. A Twitter user might share her text with another party that she trusts but this user may not know how her information will be redistributed on the Internet. The user might also not realize how much private information she is exposing. In such cases, understanding how risky other users are by assigning a privacy score to those users’ timelines can help a user decide how much sensitive information she is willing to share with users of certain privacy scores. In order to study and understand privacy behaviors in aggregate, especially as they are embedded in social networks, ‘privacy detective’ can attribute a privacy score to a Twitter timeline using a learning based approach.

Privacy varies from individual to individual and each user may have differing views of privacy. Nonetheless, there is an imperfect and non-negligible societal consensus that certain material is more private than other material in the general societal view. This societal consensus can be captured by having AMT workers annotate tweets as private or not according to Table 3.2 to calculate the privacy scores of Twitter users.

Privacy scores within a user’s network could be used to understand how social interactions influence users’ privacy behaviors. A reliable method can associate users to privacy levels to analyze how privacy behavior is influenced. Do the people a user follows or mentions in tweets influence her sensitive information-sharing behavior? Does the number of followers a user has affect her privacy habits? The proposed method ‘privacy detective’ can classify Twitter users’ timelines according to the amount of private information being exposed and associate each user with a privacy score.

Outliers in timelines are important since a privacy preserving user can all of a sudden decide to reveal a very rare disease or homeland security information. ‘Privacy detective’ is not trying to catch such extreme cases and it is not designed for self censoring. Such outliers do not have an adverse effect on collective privacy behavior.
analysis, since the focus of the study is on population level effects.

The hypothesis here may simply be stated as, those who follow or reply to users who frequently divulge private information are at a higher risk for having their private information exposed. For example, the user may release private information directly, or the release of private information may occur by an encouragement effect in which a user replies to a post from another user revealing private information which they would not have otherwise posted publicly. Intuitively, certain users will be more likely to reveal private information. Are users are more likely to reveal private information on their own, or by the influence of their friends, or after prompting from another user?

The benefit for a user having the ability to detect this type of effect is twofold. First, if users have a measure of the full extent of their contacts’ release of private information they may take steps to safeguard themselves. Second, if there is a relationship between users providing private information in replies, users of these types of systems will be more aware of the risks in such situations.

‘Privacy detective’ can uncover new things about aggregate privacy behavior. The loss of privacy has become prevalent as online social networks expand and privacy behaviors seem to be socially constructed. A quantitative analysis of the extent of the user-to-user influence in sensitive information revealing habits can demonstrate a possible factor that is contributing to the loss of personal and online privacy. This analysis can be used to improve privacy enhancing technologies and educational interventions. For example, a user can apply this on friends’ status messages to get a sense of their privacy scores and build friends lists accordingly.

Privacy behavior analysis has been influenced by the study on the collective dynamics of smoking in a large social network [34] and the spread of obesity in a large social network over 32 years [33]. Christakis and Fowler used network analytic methods
and statistical models to derive results from these studies. They examined whether weight gain in one person was associated with weight gain in her friends, siblings, spouse, and neighbors. They concluded that obesity appears to spread through social ties. They also examined the extent of the person-to-person spread of smoking behavior and the extent to which groups of widely connected people quit together. They concluded that network phenomena is relevant to smoking behavior and smoking cessation. These findings had implications for clinical and public health interventions to reduce and prevent smoking and to stop the spread of obesity.

‘Privacy detective’ detects the presence and amount of private content given text input using topic modeling, a privacy ontology, named entity recognition, and sentiment analysis. Tweets are preprocessed to make better use of natural language processing techniques. This preprocessing is important given the source text of tweets, as Twitter has evolved a language which is challenging for natural language processing tasks. Latent Dirichlet Allocation method by Blei et al. [18] is used for topic modeling. The privacy ontology is based on the privacy dictionary contributed by Gill et al. [51]. Named entities consist of names, location, date, time, organization, money, and percentage. Sentiment analysis classifies sentences as either private or not private. Private information can fall under one or more of the following 9 categories: location, medical, drug/alcohol, emotion, personal attacks, stereotyping, family or other associations, personal details, and personally identifiable information. Features are extracted from tweets with the mentioned techniques to train machine learning classifiers on various timelines with varying degrees of privacy in order to come up with a privacy score for a user’s timeline of unknown privacy score.

The learning based approach ‘privacy detective’ is the key contribution for three reasons:

1. Privacy detective detects a broad range of privacy categories. Previous work
focuses on certain types of privacy such as location privacy, medical privacy, or writing under the influence.

2. Privacy detective adopts a learning based approach whereas previous methods focus on keyword and regular expression based detection.

3. Privacy is socially constructed and this is demonstrated by the positive correlation between a user’s and her friends’ privacy scores.

Detecting private information is a hot topic since a lot of personal information is being exposed online. It is difficult to manage private information and friends lists on various social media sites such as Twitter, Facebook, and Google+, which are frequently changing their privacy policies and, at times, sensitive information is being redistributed without the owner’s knowledge. ‘Privacy detective’ can be adapted to assist users in privacy preferences about friends lists, sharing choices, and exposed content. ‘Privacy detective’ also presents an invaluable research platform for privacy researchers since it makes it possible to study how private information is revealed over time, what affects sensitive information sharing habits, and where people expose personal information.

Text preprocessing, topic modeling, privacy ontology, named entity recognition, and sentiment analysis will be explained in detail in section 3.6.

3.2 Related Work

Mao et al. [73] study privacy leaks on Twitter by automatically detecting vacation plans, tweeting under the influence of alcohol, and revealing medical conditions. Their study focuses on analyzing these three specific privacy topics by creating filters to analyze content and automatically categorizing tweets into the three categories. They investigate who divulges information. Their study is followed by a cross cultural study
that detects these three types of privacy leaks in the US, UK, and Singapore. They discuss how their classification system can be used as a defensive mechanism to alert users of potential privacy leaks.

Sleeper et al. [106] survey 1,221 Twitter users on AMT and discover that users mostly regret messages that are critical of others, cathartic/expressive, or reveal too much information. They also show that regrets on Twitter reached broader audiences and were repaired more slowly compared to in-person regrets. The privacy categories, explained in Table 3.2, were partly influenced by Sleeper et al.’s Twitter regret categories, which are: blunder, direct attack, group reference, direct criticism, reveal/explain too much, agreement changed, expressive/catharsis, lie, implied criticism, and behavioral edict.

Wang et al. [115] survey 569 American Facebook users to investigate regrets associated with posts on Facebook. They show that regrets on Facebook revolved around topics with strong sentiment, lies, and secrets, which all have subcategories. Privacy categories used in our annotations were also partly influenced by Wang et al.’s regret list. Their survey results revealed several causes of posting regrettable content. They report how regret incidents had serious implications such as job loss or breaking up relationships. They also discuss how regrets can be avoided in online social networks.

Thomas et al. [111] explore multi-party privacy risks in social networks. They specifically analyze Facebook to identify scenarios where conflicting privacy settings between friends reveals information that at least one user intended to remain private. This paper shows how private information can be spread unwillingly when a risky user in the network gets access to other users’ personal information. To mitigate this threat, they present a proof of concept application built into Facebook that automatically ensures mutually acceptable privacy restrictions enforced on group content.
Cristofaro et al. [37] present a privacy preserving service for Twitter called ‘Hummingbird’. Hummingbird is a variant of Twitter that protects tweet contents, hashtags, and follower interests from the potentially prying eyes of the centralized server. It provides private fine grained authorization of followers and privacy for followers. Hummingbird preserves the central server to guarantee availability but the server learns minimal information about users.

Hart et al. [55] classify enterprise level documents as either sensitive or non-sensitive with automatic text classification algorithms to improve data loss prevention. They introduce a novel training strategy, supplement and adjust, to create an enterprise level classifier. They evaluate their algorithm on confidential documents published on Wikileaks and other archives and get a very low false negative and false discovery rate. A support vector machine with a linear kernel performs the best on their test corpora. Their best feature space across all corpora is unigrams such as single words with binary weights. They eliminate stop words and the number of features is limited to 20,000.

Liu et al. [70] propose a framework for computing privacy scores for users in online social networks based on sensitivity and visibility of private information. The privacy score in this study indicates the user’s potential risk caused by her participation in the network.

Chow et al. [32] design a text revision assistant that detects sensitive information in text and gives suggestions to sanitize sentences. Their method involves querying the Internet for detections and recommendations.

There have been numerous studies on topic modeling [68], named entity recognition [100], and sentiment analysis [19] on Twitter as well as normalizing micro-text [118] though not focusing on tweets in particular.
3.3 Problem Statement and Threat Model

The main problem we investigate in this work is: ‘Does the given text contain any private or sensitive information and if it does, how much of the text reveals private content?’ We want to control the type of information we reveal in our text that is submitted online. We also want to know the private information sharing habits of people in our network in order to make sharing decisions based on their privacy scores. This also helps us understand social influences for revealing private information. Detecting private information is crucial for analyzing textual content and privacy behavior embedded in social networks.

The assumption in the worst case is that, an adversary will have access to all content posted by a user to the social network. Any publicly posted information may be captured by an adversary who is constantly monitoring public portions of the social network. Twitter feeds that are being analyzed are either entirely public or private, and thus the adversary can focus on users with knowledge that she has captured their full set of activity. The adversary does not have supplemental information to associate with each particular user that is not available through the Twitter system.

Users’ social behavior can impact privacy. An online social network member Alice may be influenced by her friends to release more information than she might otherwise and then some third party observer Bob, who might be an advertiser, a potential employer, or a social enemy, uses this information to harm or embarrass her.

3.4 Data Collection

Twitter users and posts in this study are randomly selected primarily due to the open nature of the posts on that social network. Both the relationships between users and their activity on the social network are recorded. Furthermore, on Twitter,
unlike a social network such as Facebook or LinkedIn, users do not have an array of built-in fields or requests for personal data. For example, on Facebook, users are routinely requested to divulge further information to the social network which may include private information such as organizational association, current location, and specific relationship information. Twitter simply requests a username and, optionally, a location. Thus, any private information found within the service is likely to be shared without prompting from the service itself.

The process of data collection emphasizes collection of a continuous stream of a conversation on Twitter. The result of this approach is that tweets of users that are more than a single degree away from the initial user are collected and considered. In doing so, the complete chain of a conversation is captured, which may have led to the release of private information.

Each tweet is analyzed for metadata within the content of the message. This metadata includes both hashtags and user references. By associating hashtags directly to tweets, users that tweet similar content are grouped. These users might not be connected by a following-type relationship.

Experimental data collection begins with the program running on a selected seed user. The program selects up to 1,000 followers of the seed user and downloads the tweets for each of these followers. For any tweet which is in reply to another tweet, the program also downloads the originating tweets. The program iterates until it reaches the initial originating tweet. The initial originating tweet is a tweet that has been replied to, but is not a reply to any other tweet. Due to time delays with the Twitter API, this process is time consuming. Thus the automated process developed was essential in data collection.

All tweet data was collected over a period of approximately three weeks in November 2013. Twitter does not present demographic information on its users, thus it is
difficult to predict age and gender. Although Twitter permits users to enter location information, many users do not, and such information has not been used in this study. The initial seed user is a local news sportscaster from Philadelphia, consequently the majority of users live in the Philadelphia area. Up to 200 of the most recent tweets for each user were downloaded. The data collection is designed so that it cannot impact the results because ground truth is provided by AMT annotations to represent a societal consensus which is explained in detail in section 3.5.

<table>
<thead>
<tr>
<th>Item</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>95,264</td>
</tr>
<tr>
<td>Tweet</td>
<td>426,464</td>
</tr>
<tr>
<td>Follower Relationships</td>
<td>4,620</td>
</tr>
<tr>
<td>Referenced Users</td>
<td>19,123 (not included in user)</td>
</tr>
<tr>
<td>Unique Hashtags</td>
<td>180,186</td>
</tr>
</tbody>
</table>

Table 3.1: Data Set Information

Data is stored in an SQL database for easier access following collection. A Java API for accessing the data and performing queries was also developed. Table 3.1 illustrates the total number of entities captured for the data set. Due to delays caused by the Twitter API, the program was unable to collect the complete set of tweets for all followed users in a reasonable amount of time. Thus, one of the intermediate goals is to determine if there is a minimum tweet count which will give a significant chance of evaluating the likelihood of a user releasing or encouraging the release of private information.
3.5 Amazon Mechanical Turk Annotations

The Amazon Mechanical Turk (AMT) is a crowdsourcing Internet marketplace that enables individuals and businesses to use human intelligence for tasks that computers cannot currently accurately perform. The goal of AMT annotations is to obtain ground truth about how much private information Twitter users reveal. Turkers annotate the publicly available Twitter data which is used for calculating the privacy scores of Twitter users. These scores are later used in supervised machine learning to classify timelines based on privacy scores. AMT is used only for annotation purposes on data that’s publicly available.

AMT masters labeled each tweet in a user’s timeline as private or not according to Table-3.2. AMT masters labeled a total of 270 randomly selected timelines each with 500 words of tweets. 500 words of tweets were sufficient to generate accurate topic ratios in topic modeling.

AMT masters achieve the ‘master’ distinction by completing work requests with a high degree of accuracy across a variety of AMT requesters. AMT masters that have demonstrated accuracy in data categorization labeled the tweets in this data set. There were 10 random quality check tweets in addition to users’ original tweets in the timelines. Humans have manually labeled these additional tweets’ privacy categories in advance. These additional tweets are used as an inter-annotator agreement check-point, to observe the variance in privacy category interpretations of machine learning classifiers and humans. The experiments in this study utilized the annotations of AMT workers who correctly interpreted the privacy category of 80% of the quality check tweets. If a worker did not satisfy the quality check requirements for a timeline, other work requests were submitted for that particular timeline until a worker met the quality requirements.

10 generic privacy categories guided AMT workers throughout the annotations.
The categories in Table-3.2 were influenced by related work, primarily the participant reported types of regret in ‘Twitter Regrets’ [106], and regret categories in ‘Regrets on Facebook’ [115]. The percentage of tweets that fall under one of the 9 privacy categories in Table-3.2 represent the privacy score of a user’s timeline.

- Privacy score-1: If more than 70% of the tweets are not private, the user is assigned a privacy score of 1.
- Privacy score-2: If 30% or more and less than 60% of the tweets are private, the user is assigned a privacy score of 2.
- Privacy score-3: If 60% or more of the tweets are private, the user is assigned a privacy score of 3.

Figure 3.1: AMT Annotation Results

According to this calculation, 185 users had a score of 1, 57 users had a score of 2, and 28 users had a score of 3, as shown in Figure-3.1.
<table>
<thead>
<tr>
<th>CATEGORY</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location</td>
<td>Giving out location information</td>
</tr>
<tr>
<td>Medical</td>
<td>Revealing information about someone’s medical condition.</td>
</tr>
<tr>
<td>Drug/Alcohol</td>
<td>Giving information about alcohol/drug use or revealing information under the influence.</td>
</tr>
<tr>
<td>Emotion</td>
<td>Highly emotional content, frustration, hot states, etc.</td>
</tr>
<tr>
<td>Personal Attacks</td>
<td>Critical statements directed at a person, general statements rather than specific.</td>
</tr>
<tr>
<td>Stereotyping</td>
<td>Ethnic, racial, etc stereotypical references about a group</td>
</tr>
<tr>
<td>Family/Association detail</td>
<td>Revealing information about family members, or revealing their associations, e.g. ex-partner, mother-in-law, step brother</td>
</tr>
<tr>
<td>Personal details</td>
<td>e.g., relationship status, sexual orientation, job/occupation, embarrassing or inappropriate content, reveal/explain too much</td>
</tr>
<tr>
<td>Personally Identifiable Information</td>
<td>Personally identifiable information(e.g., SSN, credit card number, home address, birthdate)</td>
</tr>
<tr>
<td>Neutral/Objective</td>
<td>Neutral or objective tweets that reveal no private or sensitive information.</td>
</tr>
</tbody>
</table>

Table 3.2: Tweet Privacy Categories

Having a tool that can detect the sensitivity of a timeline relative to the societal consensus on private information is useful and interesting, especially for population-level effects. The difference between the privacy levels of exposing having the flu and the presence of a rare disease is not weighted in the privacy score calculations. Excluding such exceptions does not have an adverse effect on the analysis since the population-level privacy revealing habits on social network users can be captured without such outliers. This approach focuses on aggregate privacy behavior which is a reflection of sensitive information revealing patterns as opposed to discovering
important secrets.

A second set of annotations were requested to measure the variance among the first set of annotations, supervised machine learning results, and this second set of annotations. Master AMT workers annotated a subset of 100 timelines from the first set of 270 work requests on AMT. The privacy scores of 100 users were calculated in the same way as the first set of annotations. According to the calculation, 75 users had a score of 1, 15 users had a score of 2, and 10 users had a score of 3. Inter-annotator agreement results are discussed in section 3.7.

3.6 Approach

We consider a supervised machine learning problem and train classifiers on timelines of users with known privacy scores of 1, 2 and 3 to predict the privacy scores of timelines of interest. We calculated the privacy scores of the users with known privacy scores based on ground truth obtained from AMT annotations. A timeline of a user with unknown privacy score is preprocesssed to normalize micro-text and after that, features are extracted to be used in machine learning. Timelines are classified with privacy scores by using AdaBoost [50] with Naive Bayes classifier as a weak learner. Test data is limited to 500 words of randomly selected tweets from each users’ timeline for the reasons explained in section 3.5. The process is shown in Figure 3.2.

Naive Bayes is a popular method to provide baseline text categorization results such as ham or spam classification. Naive Bayes can outperform support vector machines (SVM) with appropriate preprocessing. In our experiments, boosted Naive Bayes significantly outperformed sequential minimal optimization [91], a type of SVM. AdaBoost is a machine learning meta-algorithm that stands for ‘Adaptive Boosting’. AdaBoost trains one base Naive Bayes classifier at a time which is tweaked in favor of instances that were misclassified by the previous classifiers, and weights this clas-
sifier according to how useful it is in the ensemble of classifiers. As long as the base learners perform even slightly better than random chance, the boosted ensemble converges to a strong classifier by majority voting.

![Figure 3.2: Workflow](image)

3.6.1 Text Preprocessing

In general, informal communication on the Internet does not tend to follow proper English conventions such as proper sentence structure. Furthermore, such communications tend to include significant amounts of abbreviations, slang, and iconography. Since users on Twitter are restricted to 140 characters, there is an increased likelihood that such shorthand will be used. This is especially true when hashtags are considered. Since hashtags are metadata contained within the tweet itself, they are important to consider for both grouping tweets and also for the release of private
information.

Tweets contain text that is specific to Twitter and contain micro-text of slang and unstructured sentences. For example, they can include hashtags to tag a certain topic and user handles to refer to another Twitter user. The average number of words per tweet in our sample is 15 and the average number of words per sentence in our sample is 11. These properties of tweets make them challenging for topic modeling, named entity recognition, and many other common natural language processing tasks. In order to create meaningful topic models and detect present entities, we need to clean up tweets and convert the English to a more formal form.

Tweets contain slang words and hashtags that are hard to process as vocabulary words. In order to get rid of these, we replace them with cluster keywords from Twitter word clusters. We use the 1000 hierarchical Twitter word clusters from the Twitter NLP project [85], which were formed by Brown clustering [23] from 56,000,000 English tweets that had over 217,000 words. We manually reviewed the clusters and selected a keyword that describes the words in the cluster. If any of the words in the timeline were present in the clusters, we replaced that word with the cluster keyword.

After converting the words to cluster keywords, we removed non-ASCII characters to reduce non-English language and pictographic characters. User handles (e.g. @johnsmith) were replaced with the word he, URLs were replaced with the keyword URL, and misspellings were corrected based on an English dictionary. These text preprocessing steps are shown in Figure-3.3.

### 3.6.2 Feature Extraction

A list of extracted features which reflect presence of sensitive information are shown in Table-3.3. The reason behind extracting these particular features and methods used to obtain the feature values are explained one by one in the following sections.
Feature Normalization

All features used in the experiments were calculated either on a normalized scale or normalized during the classification process. The majority of classifiers calculate the distance between two points by using a distance metric. If one feature’s values fall under a broad range, then that feature will govern the distance measurements and mislead the classifier [13]. Features are normalized to fit individual samples in the same scale so that they have unit norm and contribute proportionately to classification distance calculations.

Topic Ratios

Topic models are algorithms for discovering the main themes that pervade a large and otherwise unstructured collection of documents according to the discovered themes [17]. Latent Dirichlet Allocation (LDA) [18] is used to discover topics. This model allows you to consider each document in a set of documents as a collection of topics. Topic modeling assumes that when a document is created, the topics that make up that document and their proportions are selected according to the dirichlet distribution. Then, the document is created by repeatedly selecting a topic according to its proportion and a word from the vocabulary for that topic until the document is completed. Although this is somewhat convoluted, if we estimate the posterior prob-
abilities of this process using Gibb’s sampling, we can determine the topics discussed in a set of documents and the proportion of those topics present in each document.

We use MALLET [74] to train a topic model on tweets that we collected from 27,293 Twitter users 267,026 tweets through the Twitter API. MALLET topic modeling toolkit contains an efficient and sampling-based implementation of ‘Latent Dirichlet Allocation’ [18] as well as routines for transforming text documents into numerical representations and removing stop words.

Some topics of discussion are more likely to reveal private information while other topics remain neutral privacy-wise. Following this intuition, we trained a topic model from the tweet data set and used this model to infer the topic ratios in given user timelines. Topic modeling and inferencing proved more effective on preprocessed text. We used the inferred topic ratios for each topic as a feature for machine learning.
In order to find the optimum number of topics, we divided the data into two parts: training set (90% of the data) and testing set (10% of the data). We then conducted 20 runs of LDA by changing the number of topics from 20 to 400. On each run, we built an LDA model on the training set and calculated the perplexity (Eq. 3.1) of the testing set. Perplexity of an LDA model is defined as,

\[
Perplexity(D_{Test}) = \exp \left( -\frac{\sum_{d=1}^{D} \log p(w_d|\alpha, \beta)}{\sum_{d=1}^{D} N_d} \right)
\]

(3.1)

where, \(D_{Test} = \) tweet data set,
\[\sum_{d=1}^{D} N_d = \text{total number of tokens in the tweet data set},\]
\(p(w_d|\alpha, \beta) = \) probability of an entire timeline belonging to a topic.

Lower perplexity scores represent a more robust model. We chose the number of topics as 200 since it produced the most robust model with the lowest perplexity measure.

Table 3.4 shows 6 topics that fall under private or neutral categories. We extracted top 20 terms from each topic to better assess contents of the topics.

**Privacy Dictionary Matches**

One feature used in machine learning is the number of matches between the ‘privacy dictionary’ and a user’s timeline. Since the timelines are limited to 500 words, this feature is normalized across users’ feature vectors.

‘Privacy dictionary’ [[14]] is a tool for performing automated content analysis of privacy. The privacy dictionary automates the content analysis of privacy related text. Using methods from corpus linguistics, Vasalou et al. [[14]] constructed and validated eight dictionary categories on empirical material from a wide range of privacy-sensitive contexts. They show that these dictionary categories detect privacy language patterns within a given text.
## Table 3.4: Some Private and Public Topics

<table>
<thead>
<tr>
<th>Topic</th>
<th>Top 20 terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Private: Inappropriate</td>
<td>f<em>ck bad f</em>cking sh<em>t female person i'm people inappropriate sh</em>t ass laugh appeal funny man holy fun real hell hate talking</td>
</tr>
<tr>
<td>Private: Religious</td>
<td>god love jesus life bless lord give respect man world good heart christ people day job family sex hope peace</td>
</tr>
<tr>
<td>Private: Marijuana</td>
<td>marijuana reveals legal medical law philly sam protest call pot country american story smoke white prohibition hunkie smoking horror qld</td>
</tr>
<tr>
<td>Public: Sports</td>
<td>sixers game heat tonight win season team people bynum andrew year order nba games flyers play classify ers mention night</td>
</tr>
<tr>
<td>Public: News</td>
<td>change africa climate service food news storm year jobs geez location weather job adaptation direction duce shows calls japan tornado</td>
</tr>
<tr>
<td>Public: Entertainment</td>
<td>job song music great video love rank listening watching movie channel i'm make favorite show country making talking dance cool</td>
</tr>
</tbody>
</table>

The dictionary is compatible with Linguistic Inquiry and Word Count (LIWC), a text analysis software program developed by Pennebaker et al. The privacy dictionary is useful in calculating the details on the usage of categories of words across heterogenous types of text. The eight categories for privacy-sensitive contexts are ‘Law’, ‘OpenVisible’, ‘OutcomeState’, ‘NormsRequisites’, ‘Restriction’, ‘NegativePrivacy’, ‘Intimacy’, and ‘PrivateSecret’. Each linguistic category contains words and phrases, which can be used to gain an understanding of the types of content contained within the text and in relation to other content.

### Named Entity Recognition

The more specific wording a user has, the more entities are found in text. Following this intuition, the higher the specificity is the higher the chances of revealing private
information. OpenNLP’s [1] named entity recognizer extracts the number of name, location, date, time, organization, money, and percentage entities to be used as a machine learning feature. Again, since the timelines are limited to 500 words, this feature is normalized across users’ feature vectors.

**Sentiment Analysis**

Sentiment analysis is generally used to extract subjective information in text. It can be used to infer whether the source is subjective or objective, or whether the tone is positive, negative, or neutral. Sentiment analysis helps differentiate private tweets from neutral or objective tweets. Therefore, the sentiment of interest is the state of revealing private information which can be used as a feature on a tweet by tweet basis.

A sentiment classifier trains on 9 privacy categories: location, medical, drug/alcohol, emotion, personal attacks, stereotyping, family or other associations, personal details, personally identifiable information, and a not private category that contains objective and neutral tweets. These 9 categories are influenced by related work and are explained in more detail in section 3.5. Each category contains at least 6,000 words of training data made up of manually labeled tweets that represent the privacy content. Lingpipe’s *n-gram* based sentiment classifier [3] classifies tweets in a timeline as private or not private. The number of private and not private tweets are two features used in machine learning. This feature is normalized across users because of the timeline word length limit.

**Quote, URL, Handle, Retweet, Hashtag Count**

Twitter users tend to place retweets or sentences written by others in quotes. The number of quotes and retweets in timelines is a feature that represents not private
content. The number of URLs, user handles, and hashtags also have information gain and are included as supplemental features. These features are normalized, since there is a word limit on the timelines being analyzed.

3.7 Results

The first set of AMT annotations show that 10.37% of Twitter users frequently reveal personal information (privacy score-3), 21.11% reveal some private information (privacy score-2), 68.52% tend not to reveal much private information by tweeting (privacy score-1). Twitter users need to be aware that the number of people revealing private information is a significant portion of all users and make conscious decisions when thinking of posting any text with private content.

The classifier reaches 95.45% accuracy in a two class task (users with scores of 1 and 3), and 69.63% accuracy in a three class task (users with scores of 1, 2, and 3) after performing 10-fold-cross-validation by using AdaBoost with Naive Bayes and standardizing the features on the data set obtained from AMT annotators. These results show that the extracted features represent privacy from a general standpoint instead of focusing on single privacy categories. This differentiates this work from previous efforts and makes the approach applicable to a broader range of privacy concerns. Using the Brown clusters and converting the text to a format that is more natural language processing friendly was a key element of being able to distinguish private and non-private tweets. Without these transformations, accuracy drops to 58.93% in a two class task (users with scores of 1 and 3), and 38.10% in a three class task (users with scores of 1, 2, and 3) after performing 10-fold-cross-validation by using AdaBoost with Naive Bayes, and standardizing the features on the same data set without preprocessing the text.
3.7.1 Twitter Database User Scores

The classifier that trained on the tweets from the annotated data set reached 69.63% accuracy in a 3-class supervised experiment. The timelines in this data set are not present in the Twitter database. This trained classifier classified the scores of 1,982 Twitter users that had at least 500 words of tweets in their timelines. The Twitter database experiment’s results show that 18.62% of Twitter users frequently reveal personal information, 30.52% reveal some private information, 50.86% tend not to reveal much private information by tweeting. Figure 3.4 shows a privacy map of the 1,982 users, where each node represents a user, each edge represents a following relationship, and the node colors represent privacy score where light yellow is a score of 1, orange is a score of 2, and red is a score of 3.

3.7.2 Correlation between User’s Privacy Score and User’s Friends’ Privacy Score

The privacy scores of users, and the average of privacy scores of people they follow is positively correlated. This means that the higher a user's privacy score, the higher her friends’ privacy scores are and vice versa. Spearman’s Rho was calculated to measure the direction and strength of relationship between users’ and their friends’ privacy scores. Spearman’s Rho used the privacy scores of 45 users, who had at least 30 friends with sufficient amount of tweets, and these friends’ privacy scores to calculate the correlation. The resulting R value is 0.41, and two-tailed P value is 0.005, which shows that there is a statistically significant positive correlation between the two variables.

Spearman’s correlation was preferred instead of Pearson’s correlation because Spearman’s correlation does not make any assumptions about the distribution of the values, and the calculations are based on ranks, not the actual values. Pearson’s
Figure 3.4: Twitter Privacy Map
correlation assumes that both of the two variables are sampled from populations that follow a Gaussian distribution. There has been no study showing that Twitter privacy scores follow a Gaussian distribution and our sample size is not large enough to support or neglect such an argument. Three random users with privacy scores 1, 2, and 3 and their friends’ scores, are illustrated in Figure 3.5, 3.6, and 3.7. The correlation between the user’s privacy score and her friends’ privacy scores are shown by the main node’s color of light yellow, orange, or red being more dominant than the data set’s average distribution.
3.7.3 Correlation between User’s Privacy Score and Mentioned Users’ Privacy Score

There is a positive correlation between a user’s privacy score and the privacy scores of users she mentions in tweets. Spearman’s Rho calculation on 45 users that mentioned at least 30 other users with calculated privacy scores returned an R value of 0.37 and a two-tailed P value of 0.01, which shows that the positive correlation between two variables is statistically significant. This correlation is weaker than the correlation between a user’s privacy score and the privacy scores of her friends. This indicates that users prefer to follow other users that have similar privacy revealing habits and users tend to mention users with similar private information revealing habits. Nevertheless, a user’s friends’ average privacy score is a stronger indicator of a user’s own privacy score than the average privacy score of people a user mentions in tweets.

3.7.4 Correlation between User’s Privacy Score and Number of Followers

Number of followers for each user that had a calculated privacy score was obtained. There was no statistically significant correlation between a user’s privacy score and her number of followers. Both Spearman’s Rho and Pearson’s correlation coefficient
were close to 0.

For example, at the time of gathering data from Twitter, rogerfederer, who is a professional tennis player ranked world no. 4 had around 1,500,000 followers and a privacy score of 1, whereas mark_wahlberg who is an American actor also had around 1,500,000 followers and a privacy score of 3. There is no correlation between how much private information a user reveals and how many followers the user has.

3.7.5 Inter-Annotator Agreement

Cohen’s Kappa coefficient was calculated to measure the inter-annotator agreement in a 95% confidence interval. Cohen’s kappa coefficient is a statistical measure of inter-annotator agreement for categorical items which takes into account the agreement occurring by chance. Cohen’s Kappa is a measurement of concordance that can be applied to data that is not normally distributed or binary data such as true/false, but is best suited to an ordinal scale, such as the 3 point privacy score scale. Kappa statistics is generally thought to be a more robust measure than simple percent agreement calculation since it excludes the agreement expected from random chance.

Cohen’s Kappa can be calculated in two ways, namely weighted kappa coefficient and unweighted kappa coefficient. Weighted Kappa coefficient is recommended when the score categories are more than two and not binary. Weighted Kappa statistics takes the distance between different categories into account. Consequently, weighted Kappa statistics offered the most accurate agreement measurements for privacy score predictions and annotations, which have 3 categories.

Landis and Koch characterized Kappa coefficient values less than 0 as indicating no agreement, 0 to 0.20 as slight agreement, 0.21 to 0.40 as fair agreement, 0.41 to 0.60 as moderate agreement, 0.61 to 0.80 as substantial agreement, and 0.81
to 1 as almost perfect agreement.

There is a *fair* agreement between the annotations of the first set and second set of AMT annotators. The agreement between the first set of AMT annotators and the classifier is *fair*. There is also a *fair* agreement between the annotations of the second set of AMT annotators and the supervised machine learning classifications. These three results suggest that the variance of privacy annotations between humans is in the same range as the variance between human annotators and supervised machine learning classifications. Determining if a given tweet is private or not is subjective to an extent for AMT workers even though detailed annotation guidelines are available. Seeing that privacy detective’s results fall under the same level of subjectivity makes it more reliable in addition to the accuracy obtained from supervised experiments.

### 3.8 Limitations

The ground truth in the training set is provided by AMT workers and not the original writers of the tweets. Turkers were given a detailed explanation of how to annotate tweets and choose privacy categories, but the original author of the tweet might have a different intension in writing the tweet. This annotation strategy provides a man on the street view of privacy, therefore this limitation did not harm the approach.

The length of timelines and the number of tweets have an effect on how much private or sensitive information is released. A personal profile can be formed by investigating the writings of a person. The more text that is present the more accurate the profile will be. The quantified effect of writing length on the amount of personal information leakage is not clear. There are numerous components in text that are representative of private information or neutral data. Each component’s effect needs to be factored out in order to investigate the effect of text length. In order to keep the
length factor stable, this study is limited to 500 words of randomly selected tweets from a Twitter user's timeline.

Most tweets in a user’s timeline could be benign and a few could be very private. A sample of 500 words might only capture the neutral tweets from a user. Not including such exceptions in the analysis is not affecting the privacy score calculations adversely. Users’ habits rather than the outliers in their timelines are the focus of this study.

3.9 Discussion

Entity recognition requires proper English sentences to detect sentences and the entities within text. Tweets by nature do not resemble proper English sentences and therefore render natural language processing tasks quite challenging. Improving named entity recognition accuracy on tweets might boost private information detection performance.

Table 3.5 shows the information gain ranks of features. Not-private sentiment count is the most important feature followed by 13 topics and the rest of the non-topic related features. The information gain ratios which are close to 1% for all of the 215 features show that all features contribute proportionately and they are all important.

There are many topics that contribute to correct classification. Creating a topic model with correct number of topics and precise LDA parameters is crucial for accurate analysis. Topic discovery is more effective on a larger data set, which covers a greater range of topics and words. As the Twitter data set gets larger after collecting more tweets through the Twitter API, new topic models are periodically updated to include recent topics. 13 topics that had the highest information gain ranks among 200 topics are shown in Table 3.6.

66.66% of wrong predictions are a miss by one in privacy score and the remaining
<table>
<thead>
<tr>
<th>Feature</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not-Private Sentiment Count</td>
<td>1</td>
</tr>
<tr>
<td>13 Topics</td>
<td>14</td>
</tr>
<tr>
<td>Private Sentiment Count</td>
<td>15</td>
</tr>
<tr>
<td>122 Topics</td>
<td>137</td>
</tr>
<tr>
<td>Privacy Dictionary Matches</td>
<td>138</td>
</tr>
<tr>
<td>Percentage Entity Count</td>
<td>139</td>
</tr>
<tr>
<td>Organization Entity Count</td>
<td>140</td>
</tr>
<tr>
<td>Name Entity Count</td>
<td>141</td>
</tr>
<tr>
<td>Time Entity Count</td>
<td>142</td>
</tr>
<tr>
<td>Quote Count</td>
<td>143</td>
</tr>
<tr>
<td>Retweet Count</td>
<td>144</td>
</tr>
<tr>
<td>Handle Count</td>
<td>145</td>
</tr>
<tr>
<td>Hashtag Count</td>
<td>146</td>
</tr>
<tr>
<td>URL Count</td>
<td>147</td>
</tr>
<tr>
<td>Money Entity Count</td>
<td>148</td>
</tr>
<tr>
<td>Location Entity Count</td>
<td>149</td>
</tr>
<tr>
<td>Date Entity Count</td>
<td>150</td>
</tr>
<tr>
<td>65 Other Topics</td>
<td>215</td>
</tr>
</tbody>
</table>

Table 3.5: Information Gain
33.33% of wrong predictions are a miss by two. Many of the wrong classifications lie on classifier boundaries. For example, one timeline was misclassified as a privacy score of 1, and it actually had 30% private tweets and needed to be classified as a privacy score of 2. Such cases can be eliminated by improving the quality of extracted features.

3.9.1 Future Work

A data set made up of tweets is a challenging one for text analytics compared to formal writing. Text analytics methods will be more effective on regular writings of people. This hypothesis can be tested in the future with a data set that consists of formal writing and ground truth on private information.

Relationship between text length and the amount of personal information leakage can be quantified as more annotated data becomes available, possibly annotated by the owner of the writing. Applying privacy analytics methods to other social media to detect private content in similar but differently formatted data will be useful in understanding privacy behavior in more detail.

The text analysis software LIWC has dictionaries relevant to privacy. In future work, incorporating other related dictionaries might improve classifier performance.

In future work, understanding the causal factors behind private information disclosure could be used to effectively design privacy enhancing nudges and target educational interventions.

3.10 Conclusion

Some topics are more likely to include private information since topic ratio features help in detecting private information. Entity recognition by itself is not enough to show if private information is being revealed, but added to topic features which define
the context of the entity, it greatly increases the detection rate of private information. Keyword based private information detection helps detect private information to some extent since privacy dictionary matches feature improves the accuracy by 4%, but it is too limited to be generalized to all privacy concerns.

Incrementally improving the approach and understanding the causal factors behind private information disclosure could be used to effectively design privacy nudges. Another possible direction is to provide an assistive tool to users that can be more than a research platform for privacy researchers. For example, a user can use privacy detective to have a sense of friends privacy scores to build friends lists accordingly.

Online privacy behavior might be socially constructed. This knowledge can be used to effectively design privacy enhancing technologies and target educational interventions.
<table>
<thead>
<tr>
<th><strong>Topic</strong></th>
<th><strong>Top 20 terms</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>People</td>
<td>url mammal person girl family bad dogs boy age front man cats location dog hot lucky loves color baby cat</td>
</tr>
<tr>
<td>Sports</td>
<td>order refresh year draft round eagles games pick rank game history trade fantasy number player nfl team calls top season</td>
</tr>
<tr>
<td>Fiction</td>
<td>letters url fiction lekker met hate pack win pur funny weer rico unit moet nar kick reaction net arv heel</td>
</tr>
<tr>
<td>Fun</td>
<td>url check great love free awesome site store food today photos tips party order time songs peek design weekend clothes</td>
</tr>
<tr>
<td>Emotions</td>
<td>people bad i’m admit hate love strange make annoy play it’s makes time funny feel friends true angry matter good</td>
</tr>
<tr>
<td>Location</td>
<td>url i’m philadelphia park city mayor location york philly design box bank photo search ave citizens center opening reveals day</td>
</tr>
<tr>
<td>Discussion</td>
<td>url change follow education pregnancy loss computer propulsion cycle lbs item secret money security gas save built boxing vin personal jobs</td>
</tr>
<tr>
<td>Curse</td>
<td>fuck bad fucking female person i’m people inappropriate shit ass laugh appeal funny man holy fun real hell hate talking</td>
</tr>
<tr>
<td>News</td>
<td>url school news sports high video upper fox lines group great washington wtf today temple darby location blurred weather back</td>
</tr>
<tr>
<td>Time</td>
<td>hours number days application time years minutes order ago back url unit late day started top running today shows left</td>
</tr>
<tr>
<td>Personal</td>
<td>people life things make love time good rank i’m hard emotion don’t happen stay find person feel it’s forget change</td>
</tr>
<tr>
<td>Religious</td>
<td>god love jesus life bless lord give respect man world good heart christ people day job family sex hope peace</td>
</tr>
<tr>
<td>Family</td>
<td>bad people event family person inappropriate call problems man make time feel kids admit makes world making age good thing</td>
</tr>
</tbody>
</table>

Table 3.6: Topics with High Information Gain
4. Future Work and Conclusion

4.1 Future Directions

4.1.1 Design “Nudges” and NLP-based Privacy Protection Policies

The institutional review board at Drexel University has recently approved a user study protocol to obtain direct ground truth information on privacy from Facebook users and introduce a privacy nudge to the Facebook architecture in the form of friend lists based on privacy scores. Users will download all of their Facebook data and label the information they shared according to privacy categories. The decisions they make in the presence of the privacy nudge will be compared to their past posts to quantify the effect of the privacy nudge. The users will also record their reasons for sharing private information, which would provide insight into the causal factors behind privacy behavior. All the features used in a previous Twitter study will be adjusted to the Facebook domain as well as bringing new features from privacy related linguistic word inquiry consortium dictionaries. This study will shed light on the effect of network phenomena on privacy behavior and the effectiveness of privacy nudges.

4.1.2 De-classification and Sanitization

Developing a method to detect sensitive entities in text is one of the next steps after completing the ongoing research for this dissertation. Identifying sensitive entities might be achieved by improving named entity recognition on specific domains through the use of word vector representations and topic modeling. Generalizing or swapping sensitive entities that have been detected is a potential sanitization method. Improving the state-of-the-art in natural language processing might benefit the security and privacy community, and many researchers, through de-classification, anonymization,
and sanitization.

Sanitizing text that contains sensitive and private information is of interest to the public and agencies that try to make documents public for several reasons. The first reason is to enable researchers share private research data with sociologists, psychologists, computer scientists, or linguists, without breaching subjects’ privacy. It can also be used in companies to prevent data loss. Another reason is to help detect classified documents in over-classified archives and redact sensitive data automatically to prepare the documents for publication. The Clinton administration had 33,000,000 emails that needed to be released to the public. Obama administration had 250,000,000 emails in 2010 that came to the legal custody of the national archives. Presidential library archivists systematically and manually redact pages one by one. A portion of Reagan’s library is read because archivists have not finished manually processing all of them yet. Manually processed documents can be searched through manual means, which is not practical. This manual approach does not scale anymore and urgently needs the help of an automated redaction tool to minimize the manual efforts.

4.1.3 Source Code and Binaries

De-anonymizing code authored by multiple programmers is an open research area. Open source repositories encourage collaborative programming. Consequently code authored by multiple authors is widely available. Being able to identify all the programmers in source code implemented in collaboration has practical uses. One anonymous account could also be used by multiple programmers. Identifying multiple programmers can aid in understanding who spreads malicious code or introduces vulnerabilities to repositories.

Another open research question in this area is analyzing authorship in binaries,
the common way that malware spreads. Binary analysis can focus on eliminating
the compiler effect to isolate coding style in the binary to perform malware family
classification, which is directly applicable to security problems.

4.2 Conclusion

Machine learning along with natural language processing methods are crucial for
identifying certain privacy and security issues at this big data era. Being able to
identify problems with these methods comes with an advantage. Usually, addressing
problems with countermeasures requires using similar machine learning and natural
language processing methods. These methods can quantify individuals’ stylistic fin-
gerprints found in any form of textual data. Fingerprints in writing style, in the form
of machine learning features, can be used to de-anonymize authors of anonymous
documents. Similarly, a numeric representation of coding style in source code can
be used to de-anonymize programmers. Stylistic fingerprints can be extracted from
source code, translated text, slang, and highly unstructured text. Stylometric analysis
can also link multiple identities of users across and within different platforms. Despite
all these, staying anonymous is possible. The stylistic features that de-anonymize in-
dividuals are the ones that need to be modified to anonymize style. Anonymization
is the process of rendering stylistic fingerprints insignificant. Anonymization can be
achieved with machine learning and natural language processing methods which are
similar to the ones de-anonymization utilizes. De-anonymizing authors enhance se-
curity by detecting the identities of people who perform malicious activities. In turn,
de-anonymization methods aid forensic experts and law enforcement. On the other
hand, de-anonymizing authors is a serious threat to privacy, especially for individuals
who would like to remain anonymous. Nevertheless, developing privacy enhancing
technologies first and foremost depend on identifying privacy infringing techniques.
High level information in text can be extracted with natural language processing techniques. The locations a person visits, people she meets, the time she does her activities can all be automatically extracted. A machine learning classifier can use the extracted information about topics of discussion, named entities, and semantics to characterize the privacy behavior of people in social networks. Aspects of human behavior expressed in language can be characterized and quantified. Such automated methods make it possible for researchers to study privacy behavior on a large scale. Understanding privacy behavior is a starting point for developing mechanisms that preserve privacy.
Bibliography


[31] **Chirgwin, R.** Your anonymous code contributions probably aren’t: boffins. The Register, January 2015.


[36] **Cohen, J.** Weighted kappa: Nominal scale agreement provision for scaled disagreement or partial credit. *Psychological bulletin* 70, 4 (1968), 213.


[71] **Luyckx, K., and Daelemans, W.** Authorship attribution and verification with many authors and limited data. In *Proceedings of the 22nd International Conference on Computational Linguistics - Volume 1* (Stroudsburg, PA, USA, 2008), COLING ’08, Association for Computational Linguistics, pp. 513–520.


