SCALABLE MACHINE LEARNING USING APPLICATIONS
IN BIOINFORMATICS AND CYBERCRIME

Approved by:

Dr. Tyler Moore

Dr. Michael Hahsler

Dr. Margaret Dunham

Dr. Eric Larson

Dr. Sukumaran Nair

Dr. Christopher Oehmen

SCALABLE MACHINE LEARNING USING APPLICATIONS IN BIOINFORMATICS AND CYBERCRIME

A Dissertation Presented to the Graduate Faculty of the

Bobby B. Lyle School of Engineering: Department of Computer Science

Southern Methodist University

in

Partial Fulfillment of the Requirements

for the degree of

Doctor of Philosophy

with a

Major in Computer Science

by

Jake M. Drew

(B.S., The University of Texas at Tyler, 2011)
(M.S., Southern Methodist University, 2013)

December 19, 2015

ACKNOWLEDGMENTS

I dedicate this thesis to my amazing wife Victoria and to my son Nathan Drew. Life would not be worth living without their love and support. I want to thank my parents Eddy and Linda Drew for continually supporting my dreams and for keeping me focused on what is important. I look up to my older brother Dr. Eric Drew, and I am grateful for his inspiration as a successful Neurologist. He proved to me that I am capable of this. I would also like to acknowledge my other younger brothers and sisters Joseph, Christina, Emily, and Jordan. Next, I am deeply grateful to Allen and Vicky McDowell for believing in me even when I was not sure why they did. Both of you helped me to see and believe in my own potential. I am sincerely thankful for Lorin and Nellie McDowell and Darrell and Joyce Baggett. Words cannot express my gratitude for you wisdom and guidance.

Finally, I would like to pay homage to all of my mentors who are too many list individually. Certainly I would like to thank my advisers Dr. Tyler Moore and Dr. Michael Hahsler. I'm not sure how I was fortunate enough to deserve two advisers or such excellence in educational mentorship. Dr. Maggie Dunham, thank you for always looking out for me and teaching me such valuable skills which I never would have had the opportunity to be exposed to otherwise. Dominic Giannangeli, you spent over 10 years molding me into a proficient executive and teaching me how to find the diamonds in data. You laid a foundation of excellence in my professional development. Thank you. Chuck Hopf, thank you for being the wise curmudgeon that really taught me how to snatch the pebble and write the code. You are my hero for that!

Drew , Jake M.

B.S., The University of Texas at Tyler, 2011
M.S., Southern Methodist University, 2013

Scalable Machine Learning Using Applications
in Bioinformatics and Cybercrime

Advisors: Professor Tyler Moore and Professor Michael Hahsler
Doctor of Philosophy degree conferred December 19, 2015
Dissertation completed December 19, 2015

SUMMARY

This thesis contributes multiple scalable machine learning applications in the fields of bioinformatics and cybercrime. A highly parallel framework for machine learning, called the Collaborative Analytics Framework is also presented. The framework leverages shared memory to efficiently process large datasets. Applications in bioinformatics gene sequence classification are implemented. In the gene sequence classification problem, unlabeled gene sequences are matched to sequences labeled with known taxonomies. Existing alignment-based methods are inefficient in practice and must balance performance by using shorter word lengths.

Prior alignment-free methods do not scale efficiently as the number of trained sequences grows. A new alignment-free method, called Strand, is introduced. STRAND achieves as good or better accuracy than existing alignment-free methods, at improved speed and a reduced in-memory training database footprint. STRAND achieves this by exploiting a form of lossy compression called minhashing as part of an in-memory MapReduce-style framework. Strand is also applied to shotgun classification challenges for purposes of Abundance Estimation.

Scalable machine learning applications are then applied to multiple cybercrime datasets. First, a method is presented to cluster criminal websites which are loose

copies of one another. This general method is then applied to two specific cases, detecting thousands of copied Ponzi Scheme and Escrow Fraud websites. Second, a binary classifier is developed to examine search results for luxury goods to identify websites selling knock-offs. Finally, the Strand application is also used to detect various classes of malware data treating each malware's binary content as a gene sequence and successfully detecting large volumes of malware files with a high level accuracy and processing efficiency.

TABLE OF CONTENTS

CHAPTER

PUBLISHED WORK

During my research at SMU, I have published the following papers, US patent applications, posters, and Blog posts. Many of these works are related to topics covered in this thesis while others represent ancillary research and programming interests. One of my papers received a best paper award: Automatic Identification of Replicated Criminal Websites Using Consensus Clustering Methods (IWCC 2015). This also resulted in a second journal publication on this topic: Optimized Combined Clustering Methods for Finding Replicated Criminal Websites (EURASIP Journal on Information Security 2014). In addition, I have filed for three US Patent Applications during my tenure at the University. My research has been awarded first place in Computer Science at the SMU Research Fair during 2013, 2014, and 2015. I have published over 13 technical blogs on Code Project `codeproject.com` with a total of 166,783 views and an average rating of 4.91 out of 5 stars. Two of these articles (MapReduce / Map Reduction Strategies Using C#) and (Parallel Programming in C# and other Alternatives) have over 35,000 views each. I have been a featured blogger on `datasciencecentral.com, dataplumbing.com, hadoop360.com, kdnuggets.com` and the Harvard Innovation Lab's `experfy.com`. In addition, my article: "Machine Learning in Parallel with Support Vector Machines, Generalized Linear Models, and Adaptive Boosting" was the most viewed and favorited article on `kdnuggets.com` during the week of 03/23/2014.

PUBLICATIONS

Mass Compromise of IIS Shared Web Hosting for Blackhat SEO: A Case Study, (TBD), 2015.

Jake Drew and Tyler Moore, The E-Commerce Market for "Lemons": Identification and Analysis of Websites Selling Counterfeit Goods, 24th International World Wide Web Conference (WWW 2015), 2015.

Jake Drew and Tyler Moore, Optimized Combined Clustering Methods for Finding Replicated Criminal Websites, EURASIP Journal on Information Security, 2014.

Jake Drew and Michael Hahsler, Practical Applications of Locality Sensitive Hashing for Unstructured Data, Performance & Capacity 2014 by CMG, 2014.

Jake Drew and Michael Hahsler, Strand: Fast Sequence Comparison using MapReduce and Locality Sensitive Hashing, The 5th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics (ACM BCB 2014), 2014.

Jake Drew and Tyler Moore, Automatic Identification of Replicated Criminal Websites Using Consensus Clustering Methods, IEEE International Workshop on Cyber Crime (IWCC 2014), 2014.

Jake Drew and Michael Hahsler, 'Strand: Variable Length K-mer Sequence Comparison Using Map Reduce and Locality Sensitive Hashing' (poster), SIAM Conference on Data Mining (SDM 2014) Doctoral Forum, 2014.

US PATENT APPLICATIONS

Jake Drew, Collaborative Analytics Map Reduction Classification Learning Systems and Methods, USPTO 61/761,523 US20140222736 A1, 2013.

Jake Drew, Single Pass Hierarchical Agglomerative Clustering Systems and Methods, 61/777,055, 2013.

Jake Drew, Michael Hahsler and Tyler Moore, System and Method for Machine

Learning and Classifying Data, 61/825,486, 2013.

TECHNICAL BLOGS

Jake Drew, Building The Ultimate Multipurpose Gaming Workstation Server, 5 Stars, 3,211 views, 2015.

Jake Drew, Machine Learning in Parallel with Support Vector Machines, Generalized Linear Models, and Adaptive Boosting, 4.50 Stars, 14,450 views, 2014.

Jake Drew, Practical Applications of Locality Sensitive Hashing for Unstructured Data, 5 Stars, 11,271 views, 2014.

Jake Drew, Clustering Similar Images Using MapReduce Style Feature Extraction with C# and R, 4.83 Stars, 7,163 views, 2014.

Jake Drew, MapReduce / Map Reduction Strategies Using C#, 4.80 Stars, 38,803 views, 2013.

Jake Drew, Parallel Programming in C# and other Alternatives, 4.77 Stars, 35,059 views, 2013.

Jake Drew, Controlling Your Web Camera Using C#, 4.92 Stars, 25,895 views, 2013.

Jake Drew, Getting Only The Text Displayed On A Webpage Using C#, 5 Stars, 16,340 views, 2013.

Jake Drew, A New JavaScript Wrapper Library for HTML5's IndexedDB, 5 Stars, 12,997 views, 2012.

Jake Drew, Creating N-grams Using C#, 5 Stars, 8,442 views, 2013.

Jake Drew, Automating the HTML5 Manifest, jQuery, and Plugins, 5 Stars, 6,221 views, 2012.

Jake Drew, Image Thinning Using R, 5 Stars, 5,117 views, 2013.

Jake Drew, Making Footable.Editable âĂŞ Cool Things Everyone Should Know About JavaScript, jQuery, and Plugins, 5 Stars, 4,706 views, 2013.

PRESENTATIONS AND AWARDS

Presenter, Performance & Capacity 2014 by CMG, November 2014.

Presenter, The 5th ACM Conference on Bioinformatics (ACM BCB), September 2014.

Best Paper Award, IEEE International Workshop on Cyber Crime, May 2014.

Presenter, IEEE International Workshop on Cyber Crime, May 2014.

Scholarship Recipient, SIAM Data Mining Conference Doctoral Forum, April 2014.

Presenter, SIAM Data Mining Conference Doctoral Forum, April 2014.

1st Place, The IBM Great Mind Challenge: Watson Technical Edition, March 2014.

1st Place, SMU Research Fair, Computer Science, February 2013.

1st Place, SMU Research Fair, Computer Science, February 2014.

1st Place, SMU Research Fair, Computer Science, February 2015.

Summa Cum Laude, The University of Texas at Tyler, December 2011

4.0 Cumulative GPA, The University of Texas at Tyler, December 2011

Student of the Year, The University of Texas at Tyler, Engineering and Computer Science, April 2011

Presenter, Alpha Chi National Convention, April 2011

Alpha Chi Honor Society, December 2010

Chapter 1

INTRODUCTION

1.1. Motivation

The US digital universe currently doubles in size approximately every three years [73]. In fact, Hewlett Packard estimates that by the end of this decade, the digital universe will be measured in 'Brontobytes', which represent one billion Exabytes or around two quadrillion years of music [139]. Each minute, internet users send over 204 million emails, Google receives over 4 million search requests, YouTube users upload 72 hours of video, and 277,000 tweets are posted on twitter [46]. It is estimated that in 2012, only 1/2 a percent of the data in the US digital universe was analyzed [73].

The massive volumes of information generated each day currently make it impossible to monitor everything within the digital universe. However, many acknowledge that this information is critically important. Certain investment corporations such as Artemis, Mediolanum Asset Management, and Bridgewater already publicly disclose the incorporation of online information into core investment strategies, and the SEC also deemed social media outlets as an 'acceptable information dissemination medium' for material non-public information in 2013, as long as the market has been notified that the channel is being used for such a purpose [10].

Large internet corporations, such as Google, use massively parallel Machine Learning techniques for extracting valuable content from our digital universe. During 2012, Google Machine Learning models executed on as many as 970 processing cores which consumed around 61.93TB of compressed raw data and 129 billion training exam-

ples [28]. Unfortunately, such algorithms are considered trade secrets and not publicly disclosed. While open source programming languages such as R may offer many machine learning packages, as of today, only 3 out of 72 packages on the R Machine Learning and Statistical Learning home page are referenced as operating in parallel [70].

The contribution of this thesis is to address the lack of massively parallel methods for machine learning, which are currently available in the literature and public domain.

1.2. Methods and Prior Work

I describe a number of machine learning related techniques for performing feature extraction, feature preparation, training, and classification using multicore MapReduce style processing pipeline implementations. I also present parallel programming algorithms for feature space compression using techniques such as minhashing which is a form of locality sensitive hashing. These multicore MapReduce style processing pipeline implementations are applied to the fields of bioinformatics and cybercrime using both supervised and unsupervised machine learning techniques.

1.2.1. MapReduce Style Processing Pipelines

MapReduce style programs break algorithms down into map and reduce steps which represent independent units of work that can be executed using parallel processing [31, 39, 48]. Initially, input data is split into many pieces and provided to multiple instances of the mapping functions executing in parallel. The result of mapping is a key-value pair including an aggregation key and its associated value or values. The key-value pairs are redistributed using the aggregation key and then processed in parallel by multiple instances of the reduce function producing an intermediary or final result.

MapReduce is highly scalable and has been used by large companies such as Google [39] and Yahoo! [16] to successfully manage rapid growth and extremely massive data processing tasks [106]. Over the past few years, MapReduce processing has been proposed for use in many areas including: analyzing gene sequencing data [106], machine learning on multiple cores [84], and highly fault tolerant data processing systems [31, 40].

More recently, a new model of cluster computing 'Spark' reported performance gains of 10x when compared to Hadoop's MapReduce processing model [160]. In addition, the Apache Spark platform claims performance gains up to 100x over Hadoop Mapreduce when processing data in memory [2]. Spark uses map and reduce steps similar to Hadoop's MapReduce adding in-memory RDDs (Resilient Distributed Datasets) which are data partitions that can easily be replaced when lost [160].

The performance gains achieved by Spark are similar to the multicore MapReduce style processing pipeline implementations described in this research. By keeping data from multiple map and reduce steps available in memory, large gains in performance and parallelism are achieved. Typically, traditional MapReduce processing completes mapping stages by writing the mapped results to disk where reduce stage workers process the files created by the mapping stage as input.

1.2.2. Minhashing

Working with large amounts of unstructured data (e.g., text documents) has become important for many business, engineering, and scientific applications. Locality sensitive hashing systems drastically reduce the time required to perform a similarity search in high dimensional space (e.g., created by the terms in the vector space model for documents). Locality sensitive hashing also dramatically reduces the amount of data required for storage and comparison by applying probabilistic dimensionality reduction. I concentrate on the implementation of min-wise independent permutations

(MinHashing) which provides an efficient way to determine an accurate approximation of the Jaccard similarity coefficient between sets (e.g., sets of terms in documents or sets of words extracted from gene sequences) [63, 76].

The concept of locality sensitive hashing has been around for some time now with publications dating back as far as 1999 [63] exploring its use for breaking the curse of dimensionality in nearest neighbor query problems. Prior to locality sensitive hashing, the data structures used for similarity searches scaled very poorly. Without using a method for approximation such as locality sensitive hashing, searches exceeding 10 to 20 dimensions, require inspection of most records in the database similar to a brute force linear search. Companies like Google have published improved LSH algorithms [76] using a consistent weighted sampling method "where the probability of drawing identical samples for a pair of inputs is equal to their Jaccard similarity" [76]. This means that when a minhash function is used to randomly select a permutation of values from a set, the number of matching values drawn equals the Jaccard similarity of the two sets [93]. This approximation of Jaccard similarity is highly accurate and increases in accuracy as the sample size increases [93].

In word-based sequence comparison, sequences are often considered to be sets of words. A form of locality sensitive hashing called minhashing uses a family of random hash functions to generate a minhash signature for each set. Each hash function used in the family of n hash functions implement a unique permutation function, imposing an order on the set to be minhashed. Choosing the element with the minimal hash value from each of the n permutations of the set results in a signature of n elements. Typically the original set is several magnitutes larger than n resulting in a signicant reduction of the memory required for storage. From these signatures an estimate of the Jaccard similarity between two sets can be calculated [23, 93]. Minhashing has been successfully applied in numerous applications including

estimating similarity between images [24] and documents [23], document clustering on the internet [25], image retrieval [32], detecting video copies [29], and relevant news recommendations [98].

1.3. Structure and Contribution

This research is organized around the application of multicore machine learning using MapReduce style processing pipeline implementations and locality sensitive hashing data compression techniques to challenging machine learning problems in the fields of bioinformatics and cybercrime. The contribution of this thesis is to provide additional innovation and support for scalable machine learning using applications bioinformatics and cybercrime. Chapters 5 - 8 detail these primary application-driven contributions.

In Chapter 2, I present a feature extraction framework using highly parallel producer-consumer pipelines for processing very large volumes of input data. While these 'parallel pipelines' are similar to MapReduce style processing, they differ in regards to MapReduce by allowing both the 'map' and 'reduce' stages access to the same shared memory for enhanced parallelism. I also perform the parallel extraction of features from a variety of unstructured data sources such as gene sequences, text, webpages, html, and images. I consider additional techniques such as the creation of both individual and combined distance matrices for unsupervised machine learning, extracting any number of features from webpages to estimate webpage similarity, and creating both vertical and horizontal luminosity histograms in parallel for the purposes of calculating the similarity between images.

Chapter 3 builds upon the machine learning framework and features for the domain of Bioinformatics. I use the highly parallel feature extraction framework to rapidly extract *words* from varying lengths of unstructured gene sequence data and

present techniques for the compression of unstructured gene sequence data during both learning and classification processing. Edit Distance is approximated by using Jaccard similarity when determining the similarities and differences between the *words* extracted from gene sequence data. Finally, the MapReduce style parallel processing pipeline simultaneously identifies unique gene sequence words, minhashes each word to generate minhash signatures, and intersects minhash signatures to estimate Jaccard similarity for highly accurate and efficient identification of gene sequence taxonomy feature classes.

Next, I present STRAND - the Super-Threaded, Reference-Free, Alignment-Free, N-Sequence Decoder in Chapter 4. STRAND is a machine learning platform for the identification and classification of gene sequence data into any number of gene sequence taxonomy classes using these highly parallel bioinformatics feature extraction techniques. Sequence classification determines the most likely taxonomy assignment when the taxonomic origin of a gene sequence is unknown. Taxonomic identification of gene sequences is beneficial to physicians when prescribing the best treatment and medicine to cure illness, and researchers also use sequence classification for the purpose of identifying genetic defects in the human genome. In the second use case, defects become classes used during training to identify such mutations in a particular sequence. Current methods, including the state-of-the-art sequence classification method RDP, balance performance by using a shorter word length. Strand in contrast uses a much longer word length, and does so efficiently by leveraging MapReduce style processing and minhashing to divide sequence classification processing across many parallel worker threads. Strand is able to learn gene sequence taxonomies and classify new sequences approximately 20 times faster than the RDP classifier while still achieving comparable accuracy results. I compare the accuracy and performance characteristics of Strand against RDP using 16S rRNA sequence data from the RDP

training dataset and the Greengenes sequence repository.

In Chapter 5, I develop useful machine learning features for cybercrime. Utilizing website data as input, I present methods for the extraction of numerous cybercrime features to identify Ponzi scheme, Escrow Fraud, and counterfeit good websites. This includes URL-level features, webpage-level features, and website-level features which are combined into various machine learning models for the successful identification and clustering of such websites. I demonstrate the creation of large scale distance matrices and combine multiple distance matrices to identify and cluster together loose copies of replicated criminal websites. In the case of criminals committing financial fraud, we present features which identify criminals who duplicate website content across many domains only slightly changing each website's appearance while leaving the site's core criminal functionality intact. Criminals creating both Ponzi schemes and Escrow Fraud related websites often relocate, re-brand, and operate multiple scams simultaneously across many domains. Features discussed in this chapter identify repeat offenders by clustering their obfuscated website content. I also describe other useful features to identify websites selling counterfeit goods. We use these features to classify online retailers as "counterfeit" and demonstrate the effectiveness of these features in Chapter 7. Numerous novel features such as the number of currencies accepted for payment, unusually large iFrames, and the unique brand term count are combined to distinguish between legitimate and criminal online retailers.

Chapter 6, demonstrates an unsupervised machine learning approach for the identification of replicated criminal Ponzi Scheme and Escrow Fraud websites which includes the combination of distance matrices to optimize clustering performance during both supervised and unsupervised machine learning. In this research, a novel optimized combined clustering method links together replicated scam websites, even when the criminal has taken steps to hide connections. I explore automated methods to

extract key website features, including rendered text, HTML structure, file structure and screenshots. Next a process is described to automatically identify the best combination of such attributes to most accurately cluster similar websites together. To demonstrate the method's applicability to cybercrime, its performance is evaluated against two collected datasets of scam websites: fake-escrow services and high-yield investment programs (HYIPs). This method is more accurate than general purpose consensus clustering approaches, as well as approaches designed for large-scale scams such as phishing that use more extensive copying of content. The method could prove valuable to law enforcement, as it helps tackle cybercrimes that individually are too minor to investigate but collectively may cross a threshold of significance. For instance, the method identifies two distinct clusters of more than 100 fake escrow websites each. Furthermore, this approach could substantially reduce the workload for investigators as they prioritize which criminals to investigate.

Building upon machine learning features for the domain of Cybercrime, Chapter 7 presents a highly parallel machine learning implementation for the identification of criminal websites. I discuss the identification and analysis of websites selling counterfeit goods or knockoff products. This research includes utilization of the URL-level, page-level, and website-level features, as well as webpage screenshots to identify internet webpages which are likely to be selling counterfeit goods. A binary classifier is devised that predicts whether a given website is selling counterfeits by examining these automatically extracted features. We then apply the classifier to results collected between January and August 2014 finding that, overall, 32% of search results point to websites selling fakes. Using a linear regression, we find that brands with a higher street price for fakes have higher incidence of counterfeits in search results, but that brands who take active countermeasures by filing DMCA requests experience lower incidence of counterfeits in search results. We also study how the incidence of

counterfeits evolves over time, finding that the fraction of search results pointing to fakes remains remarkably stable.

Chapter 8 utilizes both the bioinformatics features and gene sequence classification software described in Chapters 3 and 4 to detect 9 different types of malware files introduced in the Kaggle Microsoft Malware Classification Challenge (BIG 2015) [79]. This chapter ties together both cybercrime and bioinformatics using many of the cybercrime feature extraction and detection techniques described in Chapters 5, 6, and 7. Chapter 2's Collaborative Analytics Framework facilitates strategic and efficient changes within the Strand application to support processing the content within any number of malware input files as if they contained gene sequence data.

Finally in Chapter 9, I end with concluding remarks and discussion on future research opportunities.

Chapter 2

THE COLLABORATIVE ANALYTICS FRAMEWORK FOR PARALLEL MACHINE LEARNING

2.1. Introduction

This chapter introduces the Collaborative Analytics Framework for Parallel Machine Learning which is a highly parallel MapReduce style processing pipeline for machine learning very large volumes of input data. I first provide a detailed discussion of current state of the art multicore development types, packages, and alternatives. This also includes multicore development design patterns, locking strategies, and a brief exploration of process level parallelism using examples in the C# programming language which the Collaborative Analytics Framework was developed in. Next, I provide an abstract overview of the the Collaborative Analytics Framework itself discussing in detail how each aspect of the framework functions. Finally, I provide additional techniques for developing frameworks for extracting features from various data sources using parallel feature extraction techniques.

Since MapReduce and parallel processing are key multicore development paradigms considered in this research, I spend additional time discussing details relevant to both topics and provide very simplistic C# example implementations. I cover this material prior to explaining the highly complex MapReduce style producer / consumer pipelines which make up the Collaborative Analytics Framework. In Chapters 3 - 7, I present actual example embodiments of these abstract frameworks for the fields of Bioinformatics and Cybercrime. The conceptual groundwork laid in this chap-

ter supports each of these subsequent chapters which provide concrete, task based implementations for each of the frameworks presented.

2.2. Primary Types of Multicore Development

Starting with low level concepts such as "bit parallelism" (i.e. 32 vs. 64-bit processing) many different types of parallelism exist including instruction level parallelism, data parallelism, SIMD (Single Instruction Multiple Data), task parallelism, and accelerators to name a few. However, in terms of multicore software development most projects can be divided into two primary categories: asymmetric multiprocessing and symmetric multiprocessing. Of course, many exceptions to this general rule exist. However, for purposes of this research, understanding the primary distinctions between AMP and SMP is critical for choosing an optimal multicore machine learning development platform and architecture.

2.2.1. Asymmetric Multiprocessing

Multicore development projects which implement asymmetric multiprocessing are typically deployed for very low-level, specialized tasks [131]. The hardware upon which asymmetric multiprocessing applications execute includes a collection of two or more CPUs possibly utilizing heterogeneous operating systems which do not typically have shared memory. AMP systems achieve high levels of data parallelism by dedicating one or more processors to handling very specific data processing tasks. Under this type of multicore development scenario, a pure C# implementation is most likely not the optimal choice. While there are third party libraries which allow C# to deploy AMP solutions [67] and the .NET 4.5 framework has included optimizations for Non-Uniform Memory Access (NUMA) architectures [109], the current effort required to deploy AMP solutions in C# is similar to possibly higher performance

implementations using a lower level language such as C++.

2.2.2. Symmetric Multiprocessing

The most common form of multicore development is symmetric multiprocessing (SMP). Under the SMP architecture, high levels of task parallelism are achieved through distribution of different applications, processes, or threads to different CPUs typically using shared memory and homogeneous operating systems [6]. The C# programming language excels primarily in rapid SMP application development offering high levels of performance and one of the largest collections of parallel classes and thread-safe data structures available.

2.3. Multicore Development Alternatives

While there are a very large number of other development alternatives in the multicore marketplace, some of the more well-known offerings include: OpenMP, OpenCL, Thread Building Block (TBB), Message Passing Interface (MPI), and Cilk++. The following sections present high-level feature overviews and comparisons for each library. Understanding these alternatives is critical for choosing the best multicore development solution.

2.3.1. OpenMP

OpenMP is the most widely accepted standard for SMP systems, it supports 3 different languages (Fortran, C, C++), and it has been implemented by many vendors [119]. OpenMP is a relatively small and simple specification, and it supports incremental parallelism [37]. A lot of research is done on OpenMP, keeping it up to date with the latest hardware developments. OpenMP is easier to program and debug than MPI, and directives can be added incrementally supporting gradual parallelization [37]. OpenMP does not support thread level control or processor affinity [126].

2.3.2. OpenCL

The Open Computing Language (OpenCL) is a lower level "close-to-silicon" multicore development library [120]. OpenCL introduces the concept of uniformity by abstracting away underlying hardware using an innovative framework for building parallel applications. "The current supported hardwares range from CPUs, GPUs, DSP (Digical Signal Processors) to mobile CPUs such as ARM." [159]. While OpenCL offers "parallel computing using all possible resources on the end system" [120], multicore development using OpenCL can be quite complex with a steep learning curve. OpenCL requires configuration of various new abstractions such as "Work Groups", "Work Items", "Host Programs", and "Kernels" to implement its concept of uniformity [120].

2.3.3. Thread Building Block

The Thread Building Block (TBB) library is Intel's alternative for multicore development. TBB supports task level parallelism with cross-platform support and scalable runtimes [126]. OpenMP and TBB are similar in regards to the fact that the concept of threads and thread pools have been abstracted away within the library. Using both multicore development alternatives, the developer simply submits tasks without concern for how individual threads or the thread pool are being managed. This approach has both advantages and disadvantages. Using C#, multicore development can be done at either level using the Thread class, Parallel class, or other available solutions such as LINQ's AsParallel() method [53].

2.3.4. Message Passing Interface

The Message Passing Interface(MPI) is a AMP multicore development solution. MPI runs on either shared or distributed memory architectures and can be used on

a wider range of problems than OpenMP [37]. Unilke the SMP libraries each MPI process has its own local variables which is favorable for avoiding the overhead of locking. In addition, distributed memory computers are less expensive than large shared memory computers [37] which can be an important factor for large scale multicore development projects. However, being a lower level implementation, MPI can be extremely difficult to code involving many low level implementation details [37]. In addition, when a distributed memory architecture is used, performance can be limited by the communication network supporting each processor.

2.3.5. Cilk++

Cilk++ is a second multicore development alternative provided by Intel for supporting lower level implementation scenarios which may not be possible using Thread Building Blocks. Development in Cilk++ is a quick and easy way to harness the power of both multicore and vector processing with the library providing support for both task and data parallelism constructs [75]. With only 3 keywords, the Cilk++ library is relatively easy to learn providing an efficient work-stealing scheduler and powerful hyperobjects which allow for lock-free programming [75].

2.4. Rapid Multicore Development Using C#

The C# suite of multicore development features distinguishes itself from other multicore development libraries such as OpenMP by offering both lower thread level programming support along with the higher level parallel programming constructs such as the C# Parallel class Parallel.For() and Parallel.ForEach() methods. In addition to a large number of multicore processing constructs, C# also includes a large variety of concurrent data structures, queues, bags, and other thread-safe collections. Using the multicore development features available in C#, common parallel pro-

gramming abstractions such as Fork-Join, Pipeline, Locking, Divide and Conquer, Work Stealing, and MapReduce can be quickly implemented while drastically reducing project development timelines when compared to other multicore development languages.

A strength of multicore development in C# is the simplicity with which parallel programming abstractions can quickly be implemented. The following section explains some of the primary types of parallel programming abstractions giving examples of how these abstractions can be implemented using the C# language.

2.4.1. Thread Safe Locking and Memory Barriers

The C# multicore development environment offers many types of locks for thread synchronization during parallel processing. The "Lock" statement can be used to protect critical sections of parallel C# code and is the most common form of barrier used for thread-safety in C#. However, C# also offers great flexibility for "lock free" parallel processing using the "Interlocked" class [47, 140]. The Interlocked class can be used for the high performance, thread-safe, and atomic increment, decrement, or exchange of variables. While developing sound lock-free processing strategies may be more challenging, the interlocked class can perform magnitudes of order faster than the expensive "Lock" operation.

In addition to the standard "Lock" command and Interlocked class, more specialized locking mechanisms are available in C#. The SpinLock class is much faster than a regular lock. However, it never releases the CPU during locking and consumes more CPU resources. This lock type should be used with caution to achieve high performance in certain low-level locking situations where only one or two lines of code may require a lock. The ReaderWriter class is used for locking a resource only when data is being written and permits multiple clients to simultaneously read data when data is not being updated [110].

2.4.2. Fork Join

Using the Fork-Join pattern, various chunks of work are "forked" so that each individual chunk of work is executed asynchronously in parallel. After each asynchronous chunk of work is completed, the parallel chunks of work are then "joined" back together. For example, using this pattern, the Task.WaitAll() method can be called "joining" any number of tasks. All of the individual tasks run simultaneously, and none of them are returned to the caller until each of the individual tasks have completed. This same pattern can be accomplished in multiple ways within C#. Using tasks, parallel For and ForEach loops, or LINQ are just a few examples for executing the Fork Join pattern.

2.4.3. Parallel Pipelines

In a pipeline scenario, there is typically a producer thread managing one or more worker threads producing data. There is also a consumer thread managing one or more worker threads which consume the data being created by the producer. While a simple C# BlockingCollection provides the most basic support for exchanging data between threads in a parallel pipeline, these architectures can quickly become very complex when considering factors such as the speed differences between the producer and consumer and notifications between managing threads when production and consumption have started or completed. The BlockingCollection is typically used to manage the communication between the different producer / consumer threads in parallel pipelines. Using the BlockingCollection's GetConsumingEnumerable() method, consumers can continue to wait for new work items until the producer has notified the BlockingCollection that production has completed. In addition, bounded capacities can be set to help manage memory and resolve speed differences between producers and consumers.

The parallel pipeline can be taken one step further by executing any of the producer / consumer processes simultaneously in parallel. Furthermore, the BlockingCollection is thread-safe so no additional locking effort is required from the programmer when calling its add() method using multiple threads. Parallel pipelines can rapidly be established to perform independent processing tasks in parallel stages. As soon as one intermediate unit of parallel work has been produced, it can immediately be passed to additional downstream consumers for further processing.

2.4.4. Thread Pools and Work Stealing

In later versions of the .NET framework >= 4.0, tasks execute from the Task Scheduler using a Task Scheduler Type. The ThreadPool not only maintains a global queue, but a queue per thread where they can place their work instead of in the global queues. When threads look for work, they first look locally and then globally. If no work still exists, then threads are able to steal work from their peers [140]. While most parallel calls from LINQ and the Parallel class (Parallel.For / ForEach) manage their own thread pools, custom thread pools and task schedulers can be created when needed.

2.4.5. Parallel Divide and Conquer

The most popular example of the divide and conquer algorithm is Quick Sort which executes with a time complexity of $O(n \log n)$ using recursive calls to divide sorting work into buckets until the bucket size becomes 1, which is implicitly sorted. However, it is important to note that the Insertion Sort algorithm outperforms Quick Sort for much smaller values of n. Figure 2.1 illustrates optimized example code stopping the recursion and switching to Insertion Sort at an acceptable value for n.

Using C# QuickSort can be optimized to account for both the number of Parallel.Invoke() tasks executing at any given time and the size of n for each bucket.

```
static void QuickSort<T>(T[] data, int fromInclusive, int toExclusive) where T : IComparable<T>
{
  if (toExclusive - fromInclusive <= THRESHOLD)
    InsertionSort(data, fromInclusive, toExclusive);
  else
  {
    int pivotPos = Partition(data, fromInclusive, toExclusive);
    Parallel.Invoke(
      () => QuickSort(data, fromInclusive, pivotPos),
      () => QuickSort(data, pivotPos, toExclusive));
  }
}
```

Figure 2.1: C# Parallel Quick Sort Switching to Insertion Sort at Smaller Bucket Sizes. Code Courtesy [140].

When these thresholds are exceeded, the algorithm switches from parallel to serial execution. However, it is possible for a serial execution to recursively switch back to parallel at some point when the `CONC_LIMIT` is no longer exceeded. Figure 2.2 shows this implementation using C#.

2.4.6. MapReduce and Spark

MapReduce processing provides an innovative approach to the rapid consumption of very large and complex data processing tasks. The C# language is also very well suited for MapReduce style processing. This type of processing is described by some in the C# community as a more complex form of producer / consumer pipelines [108] which was also previously discussed in Section 2.4.3. When compared to these pipelines, MapReduce adds additional layers of error recovery, and the Map, Reduce, and combiner stages in MapReduce exchange data using intermediate files saved to disk. This allows individual stages to reside on multiple commodity hardware machines. A more simplistic parallel MapReduce style pipeline can easily be created in C# using many of the concepts and multicore development components previously presented in this chapter.

```
static int CONC_LIMIT = Environment.ProcessorCount * 2;
volatile int _invokeCalls = 0;
public void QuickSort<T>(T[] data, int fromInclusive, int toExclusive) where T : IComparable<T>
{
  if (toExclusive - fromInclusive <= THRESHOLD)
    InsertionSort(data, fromInclusive, toExclusive);
  else
  {
    int pivotPos = Partition(data, fromInclusive, toExclusive);
    if (_invokeCalls < CONC_LIMIT)
    {
      Interlocked.Increment(ref _invokeCalls);
      Parallel.Invoke(
        () => QuickSort(data, fromInclusive, pivotPos),
        () => QuickSort(data, pivotPos, toExclusive));
      Interlocked.Decrement(ref _invokeCalls);
    }
    else
    {
      QuickSort(data, fromInclusive, pivotPos);
      QuickSort(data, pivotPos, toExclusive);
    }
  }
}
```

Figure 2.2: C# Parallel Quick Sort Recurring to Serial Quick Sort Based the Number of Concurrent Tasks Executing. Code Courtesy [140].

In more complex forms, MapReduce jobs are broken into individual, independent units of work and spread across many servers, typically commodity hardware units, in order to transform a very large processing task into something that is much less complicated and easily managed by many computers connected together in a cluster. Commodity hardware machines each provide additional processors and memory for data processing when they are used together in a MapReduce cluster.

When a cluster is deployed however, additional programming complexity is introduced. Input data must be divided up (mapped) across the cluster's workers (computers) in an equal manner that still produces accurate results and easily lends itself to the aggregation of the final results (reduction). The mapping of input data to specific cluster workers is in addition to the mapping of individual units of input data work to individual processors within a single worker. Reduction across multiple cluster workers also requires additional programming complexity.

In C#, thread-safe data collections such as the ConcurrentBag, BlockingCollection, and ConcurrentDictionary can be used to exchange data between the "map" and "reduce" components of a MapReduce style pipeline. For example, consider the process of counting words in a text document. First a "chunking" function is used to read text from a file breaking all the text into smaller chunks which are "yield returned" to downstream worker threads for further processing as they are produced. Next the "map" function will divide each block of text into words in parallel. As words are identified by individual threads, they are placed into the thread-safe Blocking Collection for further downstream reduction processing. Figure 2.3 illustrates a "map" function for mapping words from file text using thread-safe collections to exchange data between the "map" and "reduce" functions.

```
public static ConcurrentBag wordBag = new ConcurrentBag();
public BlockingCollection wordChunks = new BlockingCollection(wordBag);
public ConcurrentDictionary wordStore = new ConcurrentDictionary();

public void mapWords(string fileText)
{
  Parallel.ForEach(produceWordBlocks(fileText), wordBlock =>
  {   //split the block into words
    string[] words = wordBlock.Split(' ');
    StringBuilder wordBuffer = new StringBuilder();

    //cleanup each word and map it
    foreach (string word in words)
    {   //Remove all spaces and punctuation
      foreach (char c in word)
      {
        if (char.IsLetterOrDigit(c) || c == '\'' || c == '-')
        wordBuffer.Append(c);
      }
      //Send word to the wordChunks Blocking Collection
      if (wordBuffer.Length > 0)
      {
        wordChunks.Add(wordBuffer.ToString());
        wordBuffer.Clear();
      }
    }
  });

  wordChunks.CompleteAdding();
}
```

Figure 2.3: C# Map Function for Extracting Words from Text. Available at: [49]

The "reduce" function in this process identifies unique words and keeps track of their frequencies. This is accomplished by using a thread-safe ConcurrentDictionary. The ConcurrentDictionary is a high performance collection of key-value pairs for which keys are managed using a hash table implementation which provides extremely fast lookups / access to each key's value. The ConcurrentDictionary also provides a simple addOrUpdate() method which allows users to check for a key, add the key when it does not exist, and update the key otherwise. Since we are only incrementing a counter here, the Interlocked.Increment() function is also used to ensure very efficient threadsafe updates to each counter variable. Both the "map" and "reduce" processes utilize C#'s Parallel.ForEach() function to perform MapReduce style processing in Parallel. Figure 2.4 illustrates a reduce function for counting each unique word produced by the mapping function in Figure 2.3.

```
public void reduceWords()
{
  Parallel.ForEach(wordChunks.GetConsumingEnumerable(), word =>
  {   //if the word exists, use a thread safe delegate to increment the value by 1
    //otherwise, add the word with a default value of 1
    wordStore.AddOrUpdate(word, 1, (key, oldValue) => Interlocked.Increment(ref oldValue));
  });
}
```

Figure 2.4: C# Reduce Function for Counting Unique Words. Available at: [49]

Finally, the entire MapReduce style pipeline is tied together by one master process which asynchronously executes the mapping function in the background while simultaneously performing the reduction function in the foreground to achieve the highly parallel MapReduce style pipeline. This pipeline is, in fact, more efficient in some regards to the standard MapReduce construct presented by Google [39] since the "map" and "reduce" stages both share access to the same memory. Using shared memory, this pipeline can begin "reduce" processing as soon as the first word is produced by the map function. Figure 2.5 illustrates a master process which executes the map

function in a background thread while simultaneously executing the reduce function to count unique words. To scale this pipeline across multiple processes and/or multiple commodity hardware machines, input data must may be divided into multiple partitions. However, no intermediate file exchanges are required between the map, reduce, and combiner stages.

```
public void mapReduce(string fileText)
{ //Reset the Blocking Collection, if already used
  if (wordChunks.IsAddingCompleted)
  {
    wordBag = new ConcurrentBag();
    wordChunks = new BlockingCollection(wordBag);
  }

  //Create background process to map input data to words
  System.Threading.ThreadPool.QueueUserWorkItem(delegate(object state)
  {
    mapWords(fileText);
  });

  //Reduce mapped words
  reduceWords();
}
```

Figure 2.5: C# Master Process Executing the "Map" and "Reduce" Functions [49]

The MapReduce style pipeline in Figure 2.5 resides on only a single machine using multiple processors. As a result, it is currently limited by the amount of memory and processors available to the current process where it is executing. However, input data may be divided between multiple processes executed on one or more machines.

More recently, a new model of cluster computing 'Spark' reported performance gains of 10x when compared to Hadoop's MapReduce processing model [160]. In addition, the Apache Spark platform claims performance gains up to 100x over Hadoop Mapreduce when processing data in memory [2]. Spark uses map and reduce steps similar to Hadoop's MapReduce adding in-memory RDDs (Resilient Distributed Datasets) which are data partitions that can easily be replaced when lost [160].

The performance gains achieved by Spark are similar to the multicore MapReduce style processing pipeline implementations described in Chapter 4. By keeping data

from multiple map and reduce steps available in memory, large gains in performance and parallelism are achieved. During traditional MapReduce, both the "Map" and "Reduce" stages write data outputs to disk eliminating the opportunity to enhance parallel processing by exchanging mapped items in memory immediately as they are created.

2.4.7. Process Level Parallelism

In Windows, each thread is allocated part of the process address space which is called the thread's user mode stack. Each thread's user mode stack is allocated space for things like CPU register state, scheduling priority, resource usage accounting, and "scratch storage" for things like passing function parameters, maintaining local variables, and saving function return addresses [135]. Threads also have a kernel mode stack which is used when a thread runs in kernel mode for things such as executing system calls. This allows the thread to perform "Ring 0" operations without getting page faults for accessing memory which does not belong to the parent process.

The basic kernel stack is about 24K on a 64-bit Windows machine [135], although it could be as large as 48K depending on the function it was created for. Obviously, repeatedly creating threads causes initial thread stack commits and other additional allocations of memory on the process and thread stacks which eventually become saturated resulting in memory errors. For example, Russinovich was only able to create around 55,000 threads on a 2GB stack while there should have been room for approximately 89,000 threads when considering a 24K thread stack size [135]. In addition, once threads begin using virtual memory they are no longer created and managed efficiently.

Since some of the parallel constructs in C# such as Parallel.For and Parallel.ForEach manage their own thread pools, it can be complicated or impossible to flood processors to an appropriate level using only a single process. Furthermore, dividing

parallel workloads between additional processes can drastically reduce the number of threads contending for access to the same thread-safe resources which are protected by locking, memory barriers, or wrapped within a thread-safe collection. Inter-process communication can also be very expensive and cumbersome.

However, I demonstrate that it is certainly possible to produce independent processing workers for the purposes of machine learning which require no communication between any number of dedicated processes. While there is not much literature on process parallelism using C#, my own research has proven very successful results when utilizing multiple processes within MapReduce style parallel pipelines for the purposes of machine learning. Utilizing these same concepts, it is also possible to divide work between any number of commodity hardware machines which have access to the same centralized disk space such as network attached storage (NAS). Such processing workers are highly efficient since all map and reduce stages have access to the same shared memory on a single device which greatly increases parallelism.

2.5. A Framework for Parallel Machine Learning

Machine learning involves the extraction of structured or unstructured input data from various data sources and transforming the raw data into a format suitable for performing subsequent training and/or classification processing. During training, descriptive patterns within the input data are identified using a consistently applied aggregation method, which typically collects descriptive data and statistics for data of known categories or classes. During classification, data of unknown categories or classes are used as input. The same consistently applied aggregation method identifies descriptive patterns within the input data for comparison and classification against one or more known classes within the machine learning system.

I now present a highly parallel data processing implementation for the purposes of machine learning comprising the distribution of input data using a consistently applied MapReduce style aggregation method and classification function across multiple processors and disks to perform simultaneous machine learning computations, while taking advantage of increased memory, reduced disk I/O, and the combined processing power of all participating computing devices.

Within this framework, I define two key terms: *map reduction aggregation* and $classification\ metric\ function$. Map reduction aggregation is used to describe any consistently applied aggregation method used during machine learning to extract descriptive data and possibly related statistics from raw input data for the purposes of training or classification during machine learning. For example, during text processing, words may be extracted as "descriptive data" for a document, and word frequencies may be counted as the "related statistic". The map reduction aggregation process is highly parallel and utilizes shared memory across all processing stages to enhance parallelism.

A classification metric function is used only during classification processing to make comparisons between input data and known classes within a machine learning system. The classification metric function is not always a "metric" and could be a score or any other statistic devised to make an accurate classification. For example, common classification metric functions used are Jaccard, Cosine, Manhattan, and Euclidean distance or similarity. However, the classification metric function is not limited to only distance or similarity metrics and other scores such as the $tf - idf$ (Term Frequency - Inverse Document Frequency) may be used.

The framework includes self-contained training and classification worker processes to efficiently consume input data during machine learning. Each training and classification worker utilizes shared memory during all stages of map reduction aggregation

and classification metric function processing to rapidly exchange and process intermediate data during machine learning. These self-contained training and classification workers also scale well on multiple commodity hardware machines when used in a machine learning computing cluster. Machine learning clusters avoid the use of traditional MapReduce processing overheads such as: intermediate files to exchange data between processing steps, sorting intermediate file data, or relying on expensive inter-process communication techniques to divide processing tasks between the numerous worker processes.

The Collaborative Analytics Framework was designed with each of these challenges in mind to produce a novel method for parallel machine learning when using various forms of big data as input.

2.5.1. The Collaborative Analytics Framework for Parallel Machine Learning

The Collaborative Analytics Framework for Parallel Machine Learning, subsequently referred to as the "framework", is a flexible framework for rapidly learning and classifying very large volumes of input data in parallel. It provides rapid parallel processing, learning, and classification of both structured and unstructured data, and allows programmers to create and deploy application specific map reduction aggregation methods and classification metric functions which are described in great detail in Section 2.5.2 and Section 2.5.3. The framework is implemented using the C# programming language which allows for a wide variety of multicore development alternatives and a rapid project implementation.

To guide the reader, the remainder of this chapter uses a "toy" illustration of web document de-duplication and classification using noun n-grams which are one or more consecutive nouns occurring in a document's text. This helps to compare at each stage the benefits of the framework to machine learning frameworks using traditional MapReduce style processing. The framework utilizes both training and classifica-

tion workers which take advantage of access to shared memory during all processing stages. Each worker is self-contained, and multiple workers can be clustered together to create a Collaborative Analytics Machine Learning Cluster. Training and classification workers are processes intended to execute on commodity hardware machines. Any number of training and classification worker processes may be executed on a single machine to enhance process level parallelism and utilize additional processing resources on the target machine. Multiple commodity hardware machines may also be deployed to execute any number of training and classification workers as part of a single Collaborative Analytics Machine Learning Cluster.

During traditional MapReduce, both the "Map" and "Reduce" stages write data outputs to disk eliminating the opportunity to enhance parallel processing by exchanging mapped items in memory immediately as they are created. The framework utilizes shared memory during all map reduction aggregation stages to achieve optimal parallel processing benefits while machine learning. Strand combines Map Reduction Aggregation with a Classification Metric Function to produce a novel framework for parallel machine learning.

2.5.2. Map Reduction Aggregation

The purpose of map reduction aggregation within the framework is to rapidly prepare and process input data during machine learning in parallel. I now compare map reduction aggregation to more traditional MapReduce style processing for the benefit of understanding the framework's advantages. Map reduction aggregation includes a preliminary map stage, any number of required intermediate map or combiner stages, and a reduce stage. In traditional MapReduce, a combiner stage is simply an intermediate or semi-reducer that further processes data prior to the final reduce stage [66]. In the framework, all stages required for map reduction aggregation processing are self-contained within a training or classification worker which allows each processing

stage access to the same shared memory at all times during machine learning. This is highly advantageous when compared to other traditional forms of MapReduce.

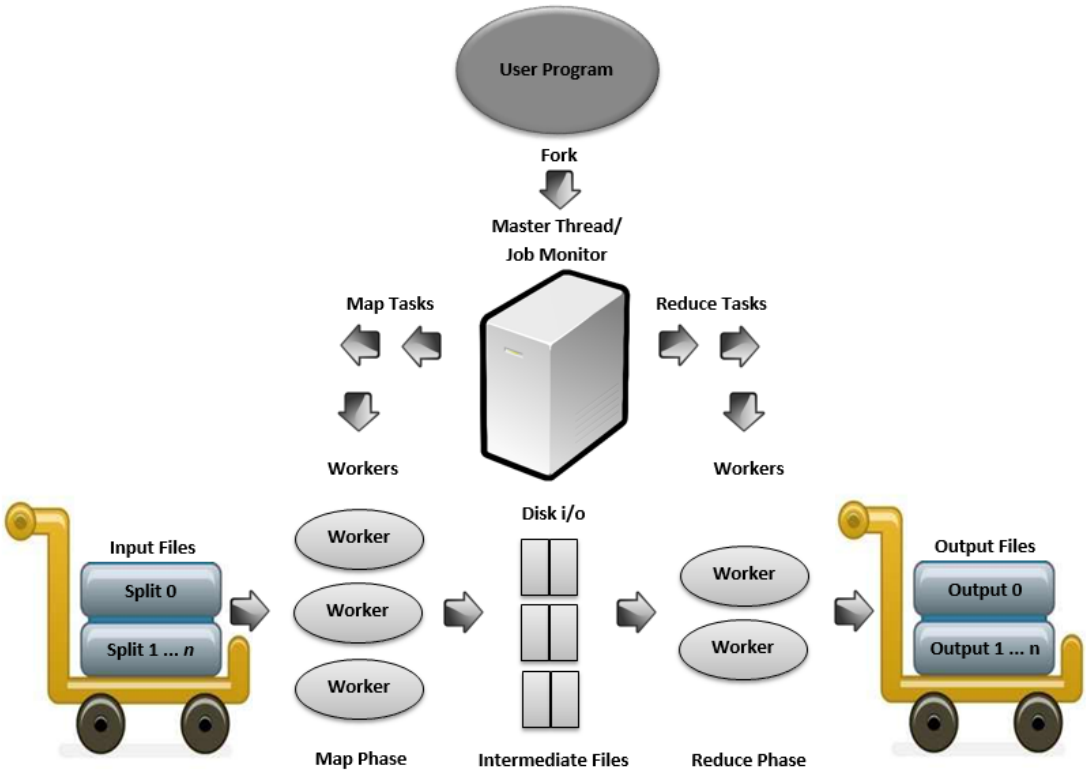


Figure 2.6: Google's Traditional MapReduce Execution Overview. Courtesy of [39]

Google's traditional MapReduce execution overview is illustrated in Figure 2.6. The following steps comprise the typical MapReduce model [39]:

1. Input data is split into multiple pieces which are managed by a master process.
2. Next, worker processes await either map or reduce tasks provided by the master.
3. Specific operations for both the map and reduce procedures are specified by the user.
4. The master monitors each map task's successful completion and notifies reduce workers of the map file output locations.

5. Intermediate files on local disks are required between each of the map, combiner, and reduce stages executed for traditional MapReduce.

6. When the reduce stage reads in mapped files from disk, the data is also sorted since a large number of keys may map to a single reduce task.

7. The reduce function processes each sorted map item according to the user specified reduce operations writing results to a separate final result file for each reduce task executed.

8. Finally, the master returns control to the calling program once all reduce steps have successfully completed.

The properties of Google's traditional MapReduce model [39] are:

1. **Fault Tolerance** - Both master and slave workers are monitored and restarted after failures.

2. **Processing Locality** - Map and reduce stages are typically processed on the same physical machine where the data resides when possible.

3. **Distributed File System** - Very large input files can be mapped across many commodity hardware machines.

4. **Scalability** - Additional commodity hardware machines can be added to a computing cluster with little effort.

The framework's map reduction aggregation methods take advantage of the simplistic parallelism constructs afforded by the MapReduce model while avoiding much of the overhead associated with intermediate file disk I/O, sorting, and inter-process communication between the master and worker processes located on different commodity hardware machines.

Within the framework, a map reduction aggregation method specifies how targeted input data will be aggregated within the current system during training and classification worker processing. Targeted input data is consistently dissected by mapping and optional combiner processes into individual, independent units of intermediate work typically comprising consistently mapped data keys and values that are conducive to simultaneous parallel reduction processing. The reduce method continually and simultaneously aggregates the mapped data keys and values by eliminating the matching keys and aggregating values consistent with the specified reduce operations for all matching keys which are encountered during reduce processing.

All map, combiner, and reduce stages are self-contained within a single user specified map reduction aggregation method allowing access to several data structures in shared memory between all processing stages. The user specified map reduction aggregation method and classification metric functions operate within any number of training and classification workers to scale as required by the user or machine learning task at hand.

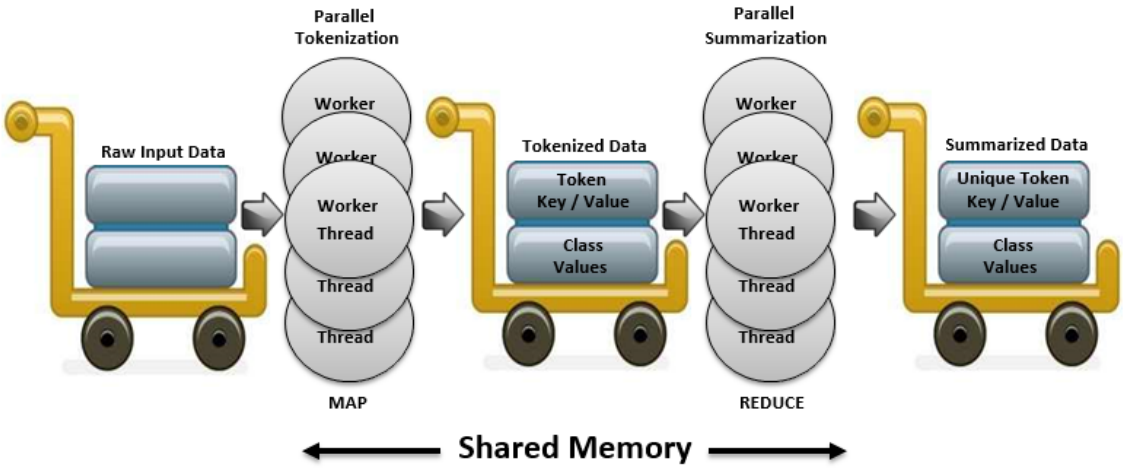


Figure 2.7: Simple Map Reduction Aggregation in the Collaborative Analytics Framework using only a Map and Reduce Stage.

Figure 2.7 illustrates a simple map reduction aggregation pipeline within the framework. Raw input data is provided to the map stage which divides the input data into keys as instructed by the map operations. The map operations may also calculate the local frequencies of keys encountered by individual worker threads during mapping. Each key and associated value is immediately passed to a thread safe data structure where the reduce stage workers simultaneously process each "mapped" item summarizing unique key values according to the specified reduce operations. Shared memory between all stages eliminates the need for any required disk I/O to store intermediate file results between processing steps. During training, one or more associated classes are passed along during each processing stage. This allows the framework to process multiple sets of input data from different classes in parallel simultaneously.

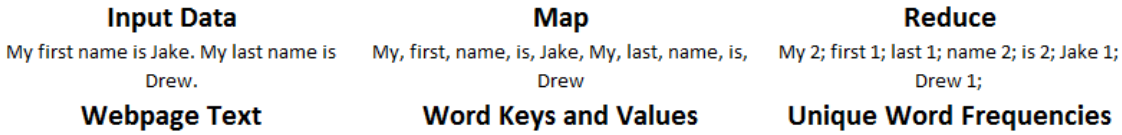


Figure 2.8: Words Extracted from Sentences using Map Reduction Aggregation.

Figure 2.8 illustrates the process for tokenizing and counting unique words displayed on a webpage using the simple map reduction aggregation pipeline shown in Figure 2.7. During the map stage, input data is split into multiple segments. Each segment is tokenized into individual words in parallel. Tokenized word keys are immediately placed into a single thread safe collection by the mapping worker threads. Using this scenario, no values are required during the map stage. However, values could be used in certain implementations to provide intermediate word frequencies within each input data split. Simultaneously, reduce stage workers are consuming each word from the thread safe collection and calculating the frequency for each unique word identified.

In the Figure 2.7 illustration, only one map stage is used prior to reduce processing. However, some map reduction aggregation methods may include multiple map, combiner, and reduce stages to facilitate increased parallelism, produce final mapping and/or reduction values, or to fulfill other requirements of specific map reduction aggregation implementations. For example, a multi-stage map reduction aggregation pipeline implementing minhashing is presented later in Section 2.5.4 and shown in Figure 2.11. Section 2.5.2.1 illustrates a multi-stage map reduction aggregation pipeline using natural language processing to extract noun phrases from web documents in Figure 2.9.

In certain framework embodiments, only keys are required during map and/or reduction processing since the value for each encountered key is always assumed to be equal to 1, or a reduction value can be calculated using only the key itself. However, in other embodiments, a value field may be necessary to maintain how many times a unique key occurred, a frequency value weighted by the key length, an aggregated series of value transformations based on a particular key's general importance within a given system, or for capturing any other value modification functions consistent with any number of application specific map reduction aggregation steps. The web document de-duplication and classification system previously described demonstrates how the framework may require multiple processing stages for a particular Collaborative Analytics machine learning embodiment. This also becomes evident when discussing locality sensitive hashing in Section 2.5.4 and the STRAND application in Chapter 4 which both demonstrate additional multi-stage map reduction aggregation pipeline scenarios. Longer keys can also be hashed to reduce the memory footprint and increase processing speed within a given system using locality sensitive hashing techniques such as minhashing during map reduction aggregation. This is discussed in detail in Section 2.5.4.

2.5.2.1. Map Reduction Aggregation for Web Document De-duplication and Classification

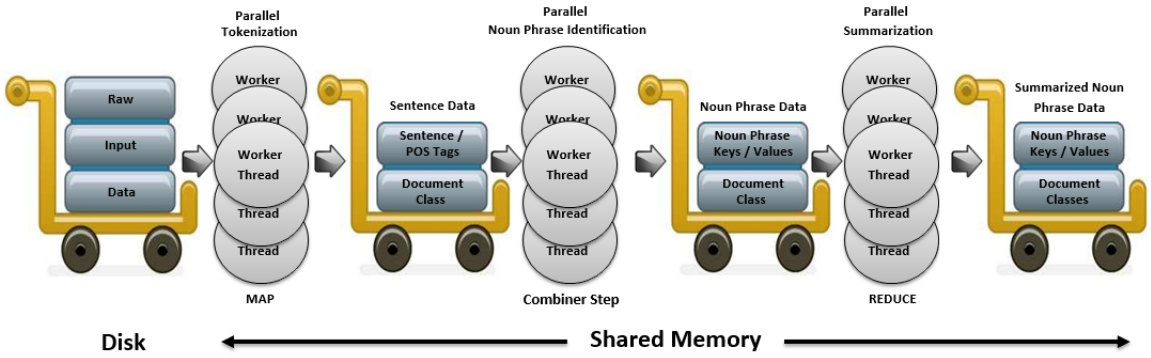


Figure 2.9: Noun n-grams Extracted from Sentences using Map Reduction Aggregation.

Figure 2.9 shows a multi-stage map reduction aggregation pipeline for extracting noun n-grams from web documents. When considering web document de-duplication and classification using noun n-grams, a typical map stage accepts the text displayed on a webpage as input mapping each webpage's text into sentences using a natural language processing engine. During mapping, multiple worker threads use the natural language processing engine to break webpage text into sentences and provide a part of speech tag for each word within each sentence. Tagged sentence keys are immediately accessible to combiner stage worker threads which identify all nouns and noun phrases within each sentence. The combiner step places each noun or noun phrase identified into a single thread safe collection accessible to the reduce step worker threads which simultaneously identify unique nouns and noun phrase keys summarizing their values concordant with the specified reduce step operations.

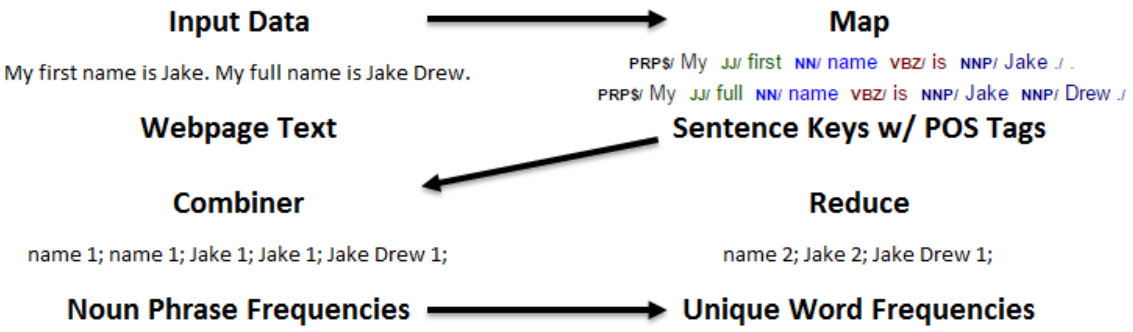


Figure 2.10: Noun Phrase Data during Map Reduction Aggregation Stages.

In Figure 2.10, input data is shown at each stage of map reduction aggregation. The map stage outputs illustrate tokenized sentences with each word token including part of speech tags provided by the SNoW tagger [65, 134]. The combiner stage identifies each word tagged as a noun (NN, NNP, NNS, NNPS) [65] capturing uni-gram, bi-gram, and tri-gram noun phrases within each sentence. In the combiner step, nouns and noun phrases appearing more than once within a sentence will have a frequency greater than 1. The reduce step, produces a list of each unique noun and noun phrase summing the frequencies of the individual combiner stage values.

During traditional MapReduce, the first map stage writes all results to disk where a combiner or reduce stage locates them as input and re-loads each map file into memory for further combiner and/or reduce stage processing. This is highly inefficient compared to the shared memory exchanges which occur within map reduction aggregation. In our web document example, the "keys" in the first map stage are sentences, and no values are required for this stage since the sentences will be processed further and divided into noun phrases. The "keys" in the combiner stage are unigram, bi-gram, and tri-gram noun phrases. The "values" in this stage may be the frequency of each unique noun or noun phrase found within a webpage document, document segment, or sentence depending on the specified combiner operations. Finally, the

"keys" for the reduce stage are each unique noun and noun phrase known to the system, and the "values" are the frequency that each unique noun and noun phrase within the system occurs. However, a more naive system may use a single map and reduce step, mapping each web document into word or letter tri-grams and using the reduce stage to count each unique tri-gram occurrence. The naive implementation avoids the considerable overhead of breaking words into sentences and identifying the part of speech for each word within each sentence during natural language processing, but it will also create many more keys during the reduce stage since no words are eliminated.

2.5.3. The Classification Metric Function

The classification metric function is a consistently applied similarity, distance, or other statistical measure used to compare two sets of map reduction aggregation outputs. These outputs could be produced by a training or classification worker, or they could be contained within the training data structure and represented as one or more known classes within the system. The training data structure is described in detail within Section 2.5.5 and illustrated in Figure 2.12.

Classification metric functions using distance or similarity metrics that only require the determination of the intersection and union between two sets, such as Jaccard similarity or distance, may only require the use of map reduction aggregation output keys, not values, during classification metric function calculations since the frequency of each word is not required. However, a single framework may still maintain both map reduction aggregation output keys and values within the training data structure for flexibility of use and in order to accommodate multiple distance or similarity metric calculations. For instance, the Cosine similarity or distance metric which uses a vector of frequencies created from the map reduction aggregation output values could also be supported as a secondary classification metric function within such a

system.

Other distance or similarity metrics may also be used which require both the keys and values produced by the map reduction aggregation process. For example, the length of each key may be used to weight each value when longer matching keys are considered to be more significant matches than shorter ones. Another variation could reference each key within a lookup table to determine a predefined key weighting used during value transformations. A detailed discussion of classification metric functions supported by a single Collaborative Analytics Training Data Structure are presented in Section 2.5.5.

Certain framework implementations may include classification metric functions which occur in tandem to map and reduce operations. While map workers are performing mapping specific operations, transitional mapping outputs are placed into centralized, thread-safe storage areas accessible to reduction operation workers. Simultaneously, reduction operation workers are consuming the transitional mapping output, reducing the keys, and integrating the values consistent with the specified reduction operations.

Simultaneously, transitional classification metric function inputs are placed by the reduction workers into the centralized, thread-safe storage areas accessible to a plurality of classification metric function workers. These workers consume the classification metric function inputs performing one or more distance and/or similarity calculations concordant with the specified classification metric function operations. Transitional classification metric function inputs may comprise matching keys between a input data query and one or more known classes including optional frequencies when required by the classification metric function.

The framework may also process input data with no known categories or classes during classification. In these instances, the classification metric function produces

a pairwise distance matrix and implements a clustering method such as k-means or hierarchical agglomerative clustering to process and cluster the transitional distance or similarity measure outputs produced. The resulting clusters can then be used as the assigned unsupervised training classes.

In the web document de-duplication and classification example, noun n-grams are extracted from the input data for either de-duplication or classification against other known web document classes within the training data structure.

In this scenario, we are answering one of two questions:

1. Has the document been seen before?

2. How similar is the document to all other known documents within the system?

Using a single training data structure, multiple classification metric functions may be supported. Jaccard Similarity is calculated between a noun n-gram query set and each noun n-gram class set within the training data structure using only the intersection divided by the union of the two noun n-gram sets. No frequency values are required for this similarity measure. The Jaccard similarity between two sets X and Y is defined as $S_J(X, Y)$, where:

$$S_J(X, Y) = \frac{|X \cap Y|}{|X \cup Y|}$$

Weighted Jaccard Similarity is also supported when the frequency values contained in the nested categorical key-value pair collection are taken into consideration [76]. The Weighted Jaccard similarity between two multisets with frequencies $\mathbf{x}$ and $\mathbf{y}$ is defined as $S_{WJ}(\boldsymbol{x}, \boldsymbol{y})$, where:

$$S_{WJ}(\boldsymbol{x}, \boldsymbol{y}) = \frac{\sum_i min(\boldsymbol{x}_i, \boldsymbol{y}_i)}{\sum_i max(\boldsymbol{x}_i, \boldsymbol{y}_i)}$$

Cosine Similarity is supported using the two frequency vectors given by the union and associated frequencies for the two noun n-gram sets. First the product for each frequency pair is summed to calculate the dot product. Next, the square of each frequency within both frequency vectors are summed to calculate the two the respective magnitudes. The Cosine Similarity for two frequency vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ is defined as $S_{cos}(\boldsymbol{x}, \boldsymbol{y})$, where:

$$S_{cos}(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{\| \boldsymbol{x} \| \| \boldsymbol{y} \|} = \frac{\sum_{i=1}^{n} \boldsymbol{x}_i \cdot \boldsymbol{y}_i}{\sqrt{\sum_{i=1}^{n} (\boldsymbol{x}_i)^2} \cdot \sqrt{\sum_{i=1}^{n} (\boldsymbol{y}_i)^2}} \tag{2.1}$$

The dot product and both magnitudes are calculated using a single pass of the union and associated frequencies for the two noun n-gram sets. Finally, Cosine Similarity is calculated using the final equation.

Now, consider the logarithmically scaled term frequency tf which contains the total number of times a particular noun n-gram n is contained in the web document $\mathcal{D}$. The logarithmically scaled term frequency is defined as $tf(n, \mathcal{D})$, where:

$$tf(n, \mathcal{D}) = 1 + \log\left(\mathcal{F}n, \mathcal{D}\right)$$

Next, consider the logarithmically scaled inverse document frequency given by the function idf. The total number of classes in the system are given by N, and the inverse document frequency is obtained by dividing the total number of classes in the system by the number of classes where a particular noun n-gram appears. The count of any noun n-gram's nested categorical key-value pair entries represents the number of web documents where a particular noun n-gram appears. A value of 1 is sometimes added to the denominator of the inverse document frequency equation to avoid division by zero for missing terms. However, the framework typically excludes terms that do not exist in the training corpus $\mathcal{C}$. The logarithmically scaled term frequency is defined as $idf(n, \mathcal{C})$, where:

$$idf(n, \mathcal{C}) = \frac{N}{|\{c \in \mathcal{C} : n \in c\}|}$$

The term n represents a particular noun n-gram within a document c while N represents the total number of documents within the system .

Finally, the Term Frequency-Inverse Document Frequency is defined as $tfidf(n, \mathcal{D}, \mathcal{C})$, where:

$$tf - idf(n, \mathcal{D}, \mathcal{C}) = tf(n, \mathcal{D}) \cdot idf(n, \mathcal{C})$$

During classification metric function processing, a summarization step is required to add up $tf - idf$ scores for each term within each class. For example, each noun n-gram within a query document would supply an individual $tf - idf$ score for each known document within a system. The classification metric function then calculates the total of all $tf - idf$ scores within each document class. The document class with the highest $tf - idf$ total is then considered most similar to the query document in question.

These are just a few examples of classification metric and scoring functions available when using only a single training data structure. Many other classification metric functions are possible when different map reduction aggregation techniques are utilized. For example, a more efficient and performance optimal classification metric function becomes available when minhash signatures are created during map reduction aggregation as discussed in Section 2.5.4.

2.5.4. Lossy Compression Using Locality Sensitive Hashing

Locality sensitive hashing is utilized within the framework to drastically reduce the amount of storage required for high capacity map reduction aggregation and classification metric function operations. Map reduction aggregation requires multiple

pipeline stages when a locality sensitive hashing engine is deployed.

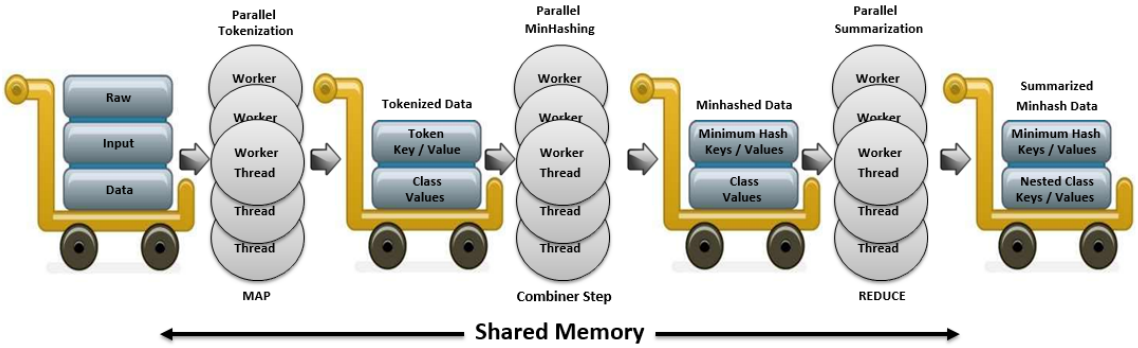


Figure 2.11: Map Reduction Aggregation using Minhashing.

In Figure 2.11, the framework uses a more complex map reduction aggregation pipeline including an additional combiner step to facilitate minhashing. Minhashing is a form of lossy data compression used to remove a majority of the keys produced during stage one mapping by selecting the smallest value produced for each of a family of random hashing functions in order to create a much smaller minhash signature.

During stage one of the map reduction aggregation method shown in Figure 2.11, transitional outputs are placed into centralized, thread-safe storage areas accessible to minhash operation workers. In stage 2, a pre-determined number of distinct hashing functions are then used to hash each unique key produced during the stage one map operation one time each. As the transitional keys are repeatedly hashed, only the minimum hash value for each of the distinct hash functions are retained across all keys. When the process is completed only one minimum hash value for each of the distinct hash functions remains in a collection of minhash values which represent the unique characteristics of the learning or classification input data within a minhash signature.

Minhash collections may also be consolidated or further reduced by storing each minhash signature in a partitioned collection of nested categorical key-value pairs.

The training data structure in Figure 2.12 is designed in this manner. The training data structure's nested key value pairs are partitioned or sharded by each distinct hash function using a distinct hash function id. For example, when the minhashing process uses 100 distinct hash functions to create minhash signatures, the training data structure is divided into 100 partitions. All unique minhash keys created by hash function 0 are stored within partition 0 of the training data structure. All unique minhash keys created by hash function 99 are stored in partition 99.

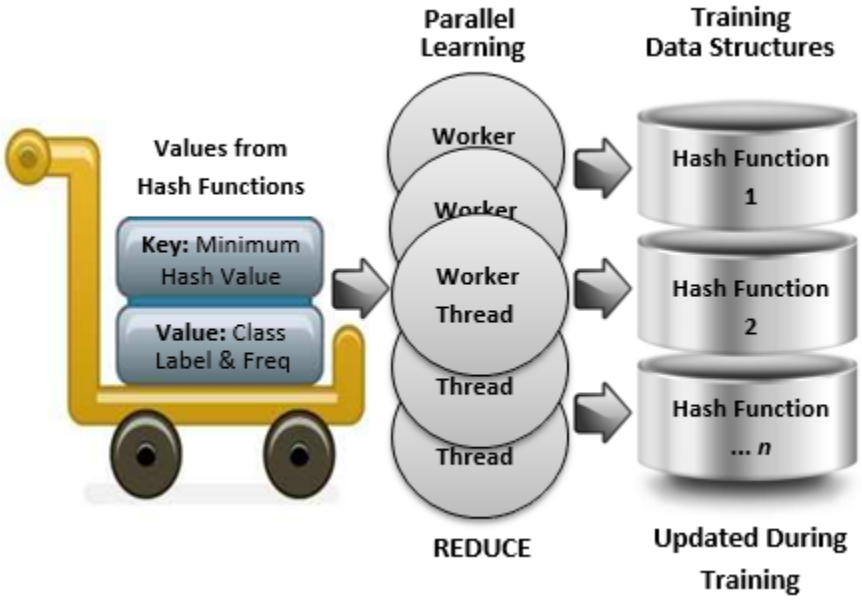


Figure 2.12: A Collaborative Analytics Partitioned Training Data Structure.

Figure 2.12 shows a partitioned framework training data structure where minimum hash values act as the "key" in the nested "categorical" key-value pair collection. Each minhash key contains as it value a collection of the classes which are associated with that key in the system. This collection of classes represents the nested categorical key-value pairs collection. Each nested categorical key-value pair contains a known class as its key and a frequency, weight, or any other numerical value which represents the importance of the association between a particular class and the minhash value.

This numerical description value is typically created by the reduction method, and the number of nested categorical key-value pair entries can also vary between the individual minhash key entries within the entire collection. The categorical key-value pair collection is nested to support a single minhash value belonging to more than one class. However, a simpler structure can be used in systems which only support one class association for each minhash key.

Map reduction aggregation could be performed on any number of input data collections and reduced into a single nested minhash categorical key-value pair collection for the purpose of performing a combined training or classification. For example, multiple training data structures are produced in parallel by multiple training workers within a Collaborative Analytics computing cluster and then consolidated. Once the minhash signature has been created, the classification metric function is applied to calculate similarity or distance measures between the input data's signature and known classes within the training data structure for the purposes of classification.

While other framework processing pipelines may use any number of classification metric functions, pipelines which utilize minhashing typically only deploy the Jaccard similarity or distance metric which is approximated by intersecting two sets of minhash signatures where longer signatures provide more accurate Jaccard similarity or distance approximations [129]. Class frequencies may be used to produce other Jaccard Index variations such as Weighted Jaccard Similarity [76]. However, large performance gains are achieved using binary classification techniques where no nested categorical frequency values or log based calculations are required during classification metric function operations. In the binary minhash classification approach, minhash signature keys are simply intersected with the minhash keys of known classes to calculate similarity. This is demonstrated with great success when classifying gene sequence input data in Chapter 4.

Using the example of web document de-duplication and classification, the total number of noun n-grams produced when processing millions of web documents quickly becomes unfeasible to store. Instead, each of the noun n-grams are now sent through a family of random hashing functions to create a minhash signature. The minhash signature represents the smallest value produced by each individual hashing function utilized. Each noun n-gram located within a webpage is hashed by each hashing function and only the minimum values for each hashing function are retained as part of the minhash signature. A webpage document with 10,000 noun n-grams may be reduced to only 100 integer minhash values when 100 hashing functions are used to create the minhash signature. Likewise, the training data structure is divided into 100 partitions dedicated to storing training data for each of the 100 unique hashing functions. This increases parallelism by reducing contention between keys being added to a single collection during training. In addition, the same key produced by two different hash functions will have two different meanings and must be accommodated. Compression rates greater than 98% are achieved using minhashing to reduce the number of words created from a single gene sequence in Chapter 4.

2.5.5. Applying The Collaborative Analytics Framework to Machine Learning Tasks

While traditional MapReduce is commonly used for multicore machine learning tasks [30, 61, 62], researchers [60, 101] now recognize the need for parallel machine learning frameworks which strike a balance between the high-level parallel abstractions like MapReduce and the low-level multicore development tools discussed in Section 2.3.

Gillick et. al. states that an ideal MapReduce style parallel machine learning implementation should provide shared memory to all map tasks on a compute worker [62]. The framework takes this idea one step further by exposing shared memory for all processing stages required to create a consistently applied data preparation

and aggregation method for the purposes of training or classification during machine learning. In addition, the framework's self-contained training and classification workers are easily replicated to scale a machine learning process on any size computer or on any number of commodity hardware machines operating within a cluster.

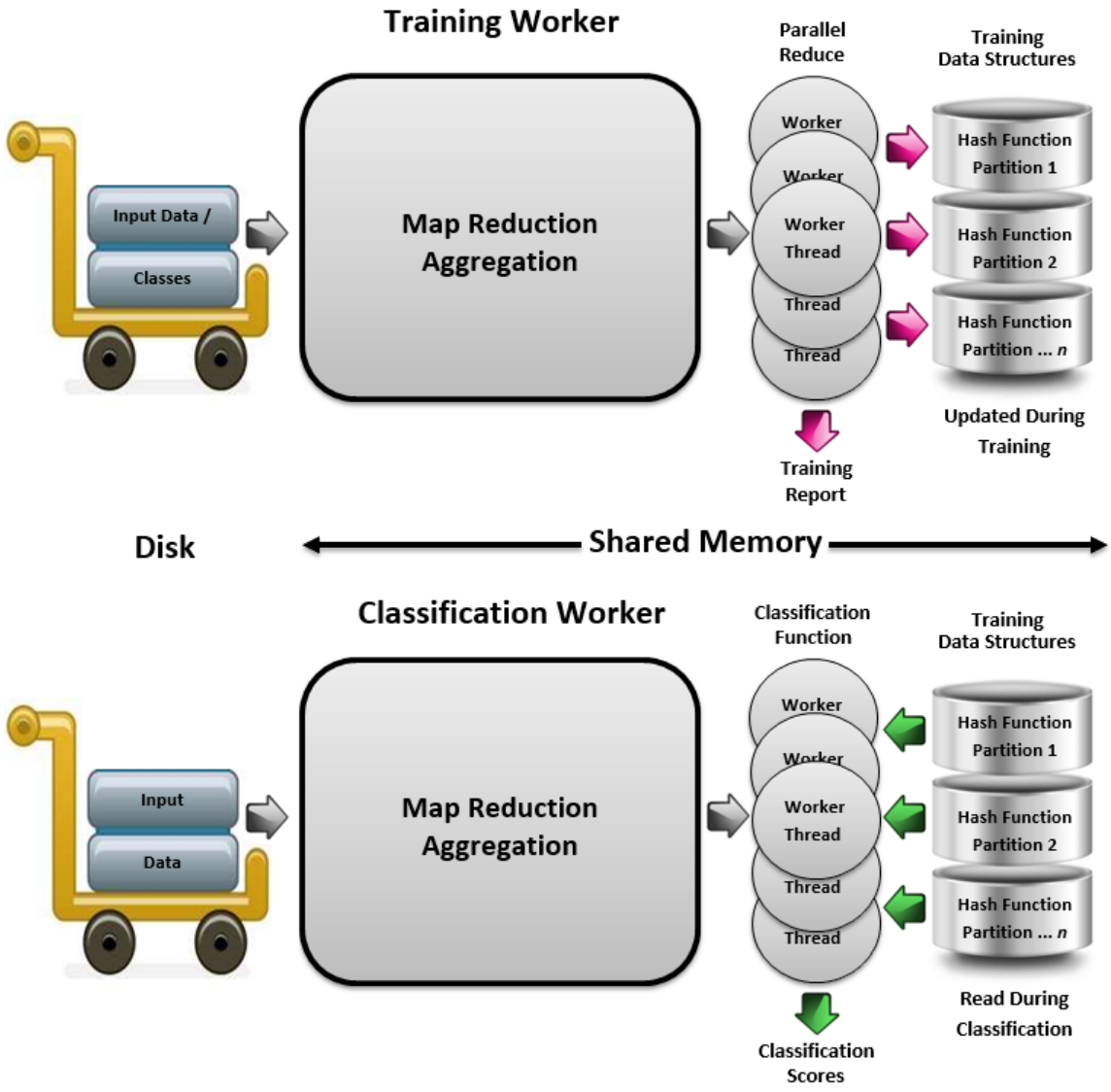


Figure 2.13: Collaborative Analytics Training and Classification Workers

In Figure 2.13, map reduction aggregation output of known classes are consolidated or further reduced by storing all outputs using the same map reduction ag-

gregation method in a single collection of nested key-value pairs where the key for each key-value pair maps to a second or nested collection of "categorical" key-value pairs as its value. This is called the training data structure. The nested collection of categorical key-value pairs for each key provides a numerical description of that key across any number of categorical keys (i.e. known classes). The training data structure may also be partitioned to support more complex map reduction aggregation outputs or to enhance parallelism during machine learning. A partitioned training data structure was also presented in Figure 2.12 for the purpose of supporting lossy compression using minhashing. The training data structure is updated during training worker processing and read during classification worker processing to apply the specified classification metric function operations.

In the web document de-duplication and classification example, unique noun n-grams are summarized during map reduction aggregation calculating the frequency of each unique noun n-gram identified. The web documents associated with each noun n-gram are made available to the reduce step and act as the associated classes in this example. In the naive approach, the training data structure contains a single partition where each unique noun n-gram key contains a nested collection of web document classes and the associated frequency for a particular noun n-gram within each web document class as the nested categorical key-value pair collection. However, when minhashing is used, an additional combiner step is added to hash each noun n-gram one time retaining the minimum hash value for each unique hashing function used. These values make up the minhash signature and are stored in the partitioned training data structure.

During training, each noun n-gram entry is looked up within the training data structure. New training data structure entries are created for new noun n-grams while the frequency value is incremented when a noun n-gram key already exists. Multiple

worker threads are operating at each stage of the machine learning pipeline in parallel. The map, combiner, and reduce stages within the map reduction aggregation stage operate simultaneously exchanging intermediate data using shared memory within each self-contained training worker. This is highly advantageous since all disk I/O is eliminated between each of the map, combiner, and reduce stages. When processing very large volumes of input data, multiple training or classification workers are deployed to reduce processing times and add additional processing hardware within a Collaborative Analytics Cluster. This is discussed further in Chapter 4.

During classification, the same map reduction aggregation method used during training processes the classification input data. However, a classification metric function also calculates the distance or similarity between the map reduction aggregation outputs and one or more classes stored within the training data structure. When making a multi-class prediction, the class with the highest similarity or the lowest distance is selected. However, the framework is also capable of providing the similarity or distance scores calculated for each of the individual classes. This is useful when input data may align with multiple classes. For example, a lengthy gene sequence may contain multiple mutation classes known by the framework.

Figure 2.13 also shows a Collaborative Analytics classification worker. All stages of map reduction aggregation appear identical to the training worker above. This is critical since accurate classifications are only made when the summarized data produced by map reduction aggregation for training and classification are created in an identical manner. Outputs produced from the final reduce step are looked up within the training data structure locating matching keys and associated class frequencies as needed. The map reduction aggregation output is processed according to the classification metric function operations in order to determine a classification score for one or more classes contained within the training data structure.

2.5.6. Managing Speed Differences Between Pipeline Producers and Consumers

The framework uses C# Blocking Collections to manage the exchange of map reduction aggregation pipeline data placed in transitional output storage areas. The blocking mechanism manages the potential differences in the production and consumption speeds between map reduction aggregation and classification metric function operations. Once the transitional blocking mechanism is notified that transitional output production has started, production workers produce transitional outputs while consumption workers consume the transitional outputs until the blocking mechanism is notified by the production process that production of all transitional outputs has completed.

Consumption workers also continue working until production has completed and all transitional outputs have been consumed. In the event that there are no transitional outputs to consume and production has not completed, the blocking mechanism allows consumer worker threads to "block" or wait until additional transitional outputs are produced. Memory consumption can also be managed within the blocking mechanism by setting a pre-determined production capacity. When the production capacity is exceeded, the blocking mechanism allows production workers to stop production and "block" or wait until additional transitional outputs are consumed and the transitional output count falls back below the pre-determined production capacity. Any framework process creating transitional outputs including map methods, reduction methods, map reduction aggregation, and classification metric functions could act as producers and/or consumers interacting with multiple blocking mechanisms participating in multiple production and consumption relationships within a particular framework pipeline.

2.6. A Framework for Parallel Feature Extraction

Concepts and techniques similar to those implemented in the Collaborative Analytics Framework can also be used to extract features in parallel for both supervised and unsupervised machine learning processes. In fact, this would be equivalent to creating a customized map reduction aggregation method used directly as input for another machine learning package, or using the classification metric function to create output other than the training data structure previously described. For example, a map reduction aggregation pipeline could be designed to extract the displayed text from web pages while distance calculations produced by a classification metric function are used to create a pairwise distance matrix which reflects the Jaccard Distance between all websites based upon bags of sentences displayed on each website.

A distance matrix is required as input for some clustering algorithms such as Hierarchical Agglomerative Clustering [45]. Yet another use case might be defining a custom map reduction aggregation method for extracting vertical and horizontal luminosity histograms from images, and creating a custom classification metric function using such image features to calculate the similarity or distance between images in order to produce a pairwise distance matrix used as input for image similarity clustering.

I describe detailed Cybercrime related parallel feature extraction implementations for creating feature distance matrices in Chapter 3. Implementations for extracting Bioinformatics related features in parallel are also covered in Chapter 5. In order to provide a more concrete example of implementing a framework for parallel feature extraction, I will now present a method for the extraction of luminosity histogram features from images.

2.6.1. Extracting Luminosity Histogram Features From Images in Parallel Using C#

The term "Luminance" is used to describe the luminous intensity per unit area of light as a photometric measure [155]. Luminance describes how much light passes through a given area and falls within given solid angle [155]. For a digital image, this is typically determined at the pixel level. Relative luminance follows this definition, but it normalizes its values to 1 or 100 for a reference white [156].

When separated into RGB components, a luminosity histogram acts as a very powerful machine learning fingerprint for an image. Since these type of histograms only evaluate the distribution and occurrence of luminosity color information, they can handle affine transformations quite well [123]. In simple terms, it very easy to separate the RGB components of an image using C#. In fact, using the "unsafe" keyword one can rapidly calculate the relative luminance of each pixel within an image [17,111]. For this project, I was able to quickly adapt the luminosity histogram feature extraction program contained within the open source Eye.Open library [111] to provide parallel feature extraction using the concepts previously described.

2.6.2. Extracting the RGB Channels

Once an image has been processed, both vertical and horizontal luminosity histograms are extracted and retained for similarity calculations between images. Figure 2.14 shows each pixel's red, green, and blue channels being isolated for use within the luminosity calculation using C#. Images can quickly be compared for similarity using the similarity function shown in Figure 2.15. This classification metric function calculates the deviation between two histograms using the weighted mean. Once both the vertical and horizontal similarity have been calculated, the maximum or average vertical / horizontal similarity is retained for each image pair to create a pairwise

image distance matrix.

```
unsafe
{
  var imagePointer1 = (byte*)bitmapData1.Scan0;

  for (var y = 0; y < height; y++)
  {
    for (var x = 0; x < width; x++)
    {
      var blu = imagePointer1[0];
      var green = imagePointer1[1];
      var red = imagePointer1[2];

      int luminosity = (byte)(((0.2126 * red) + (0.7152 * green)) + (0.0722 * blu));

      horizontalProjection[x] += luminosity;
      verticalProjection[y] += luminosity;

      imagePointer1 += 4;
    }

    imagePointer1 += bitmapData1.Stride - (bitmapData1.Width * 4);
  }
}
```

Figure 2.14: Using the RGB Channels to Calculate Luminosity for each Pixel in an Image in C#. [111]

```
private static double CalculateProjectionSimilarity(double[] source, double[] compare)
{
  if (source.Length != compare.Length)
  {
    throw new ArgumentException();
  }

  var frequencies = new Dictionary();

  //Calculate frequencies
  for (var i = 0; i < source.Length; i++)
  {
    var difference = source[i] - compare[i];
    difference = Math.Round(difference, 2);
    difference = Math.Abs(difference);

    if (frequencies.ContainsKey(difference))
      frequencies[difference] = frequencies[difference] + 1;
    else
      frequencies.Add(difference, 1);
  }

  var deviation = frequencies.Sum(value => (value.Key * value.Value));

  //Calculate "weighted mean"
  //http://en.wikipedia.org/wiki/Weighted_mean
  deviation /= source.Length;

  //Maximize scale
  deviation = (0.5 - deviation) * 2;

  return deviation;
}
```

Figure 2.15: Use RGB Channels from Image to Calculate Luminosity for each Pixel in an Image. [111]

There are many ways to extract features from images and create image fingerprints. Many people recommend other approaches such as taking 2D Haar Wavelets of each image [86]. One could easily extend our approach to to use Haar features of any size or degree. In addition, higher performance options such as using minhashing with tf-idf weighting have also been implemented [33]. Haar features have been successfully used for more complex Computer Vision tasks such as detecting faces within an image [38] using C#.

2.6.3. Creating the Distance Matrix in Parallel

The pairwise distance matrix is required as input when using some Hierarchical

```
/// Rapidly generate all required matrix matches in the background for processing.
/// (this allows a much more balanced execution of Parallel.ForEach on all matches)
/// Yeild return passes each match to parallel foreach in createPairwiseMatches()
/// as they are produced.
///

private IEnumerable generateMatches()
{
  for (int r = 1; r < filePaths.Length; r++)
  {
    //for each column less than the current row
    for (int c = 0; c < r; c++)
      yield return new PairwiseMatch(r, c);
  }
}


/// Parallel asynchronous background process to create all pairwise matches for a
/// given matrix, extracting image features one time (as required).
///

public void createPairwiseMatches(string inputDirLoc)
{
  Parallel.ForEach(generateMatches(), match =>
  {
    match.projectionR = getImageFeatures(filePaths[match.rowIndex]);
    match.projectionC = getImageFeatures(filePaths[match.colIndex]);
    MatrixMatches.Add(match);
  });

  MatrixMatches.CompleteAdding();
}
```

Figure 2.16: Stage One Mapping of all Pairwise Matches.

Agglomerative Clustering functions within R [45]. In order to create this matrix in parallel, a three stage parallel pipeline is used. First, pairwise image matches are created using a C# yield return enumeration. Each time the generateMatches() function in Figure 2.16 produces a pairwise match, processing stops and "yield returns" each match to the createPairwiseMatches() function's Parallel.ForEach loop. Figure 2.16 shows the relationship between these two functions which is executed in a background process during the distance matrix creation.

The createPairwiseMatches() function shown in Figure 2.16 above, extracts features in parallel mapping images to vertical and horizontal luminosity histograms. Furthermore, the histograms for each image are saved in a hash table for quick ref-

erence since each image's RGB features will be repeatedly matched and compared to other images. Once the match features are extracted, the match is immediately placed in a thread safe blocking collection for further downstream reduction processing. While the mapping functions shown in Figure 2.16 are executing in a background thread, parallel reduce functions simultaneously execute processing each completed match produced to calculate the similarity between all matched images.

```
/// Asynchronous background process to extract rgb projections (when needed)
/// and calculate similarity between images in parallel.
///

public void calculateDistances(string outputFileLoc)
{
  Parallel.ForEach(MatrixMatches.GetConsumingEnumerable(), matrixMatch =>
  {
    //Calculate similarity between two images and save it to the correct matrix row.
    double similarity = RgbProjector
      .CalculateSimilarity(matrixMatch.projectionR, matrixMatch.projectionC);
    double distance = 1 - similarity;
    matrixRows[matrixMatch.rowIndex][matrixMatch.colIndex] = distance;
  });
}
```

Figure 2.17: Classification Metric Function Stage

Figure 2.17 shows the calculateDistances() function which reduces image features into a single distance measure by comparing each image's vertical and horizontal luminosity histograms. Once this process has completed, a pairwise distance matrix is saved to disk which can be used as input into the R program's Hierarchical Agglomerative Clustering engine [45]. The final distance matrix contains pairwise distances for all images in the input directory. Figure 2.18 demonstrates that only the minimum required number of pairwise distances were retained in the final output.

	A	B	C	D	E	F	G	H	I	J	K	L
1	www.jakemdrew.com	C:\Users\J	C:\Users\J	C:\Users\J	C:\Users\J	C:\Users\J	C:\Users\J	C:\Users\J	C:\Users\J	C:\Users\J	C:\Users\J	C:\Users\J
2	C:\Users\Jake\Desktop\ImageClustering\WepageScreenshots\10percent-invest.com_index.png											
3	C:\Users\Jake\Desktop\Image	0.631311										
4	C:\Users\Jake\Desktop\Image	0.613904	0.902232									
5	C:\Users\Jake\Desktop\Image	0.449429	0.795265	0.549379								
6	C:\Users\Jake\Desktop\Image	0.916717	0.563644	1.076537	0.933804							
7	C:\Users\Jake\Desktop\Image	0.93031	0.642052	0.920941	0.901441	0.6599						
8	C:\Users\Jake\Desktop\Image	0.891842	0.635936	0.979329	0.852873	0.628398	0.262402					
9	C:\Users\Jake\Desktop\Image	0.79006	0.385576	1.003804	0.797187	0.404815	0.553764	0.484344				
10	C:\Users\Jake\Desktop\Image	0.707818	0.46048	1.015365	0.848959	0.396607	0.624204	0.560941	0.363544			
11	C:\Users\Jake\Desktop\Image	0.781171	0.40984	0.922683	0.871672	0.479079	0.575656	0.512002	0.4399	0.502122		
12	C:\Users\Jake\Desktop\Image	0.917287	0.558609	1.199299	1.05	0.496096	0.807207	0.703423	0.563824	0.481642	0.531441	
13	C:\Users\Jake\Desktop\Image	1.072793	0.736587	1.084855	1.018909	0.473213	0.456917	0.417417	0.501481	0.543884	0.661051	0.637808
14	C:\Users\Jake\Desktop\Image	0.783193	0.407327	0.952743	0.867618	0.480711	0.507948	0.43049	0.31037	0.365786	0.409089	0.501732
15	C:\Users\Jake\Desktop\Image	1.155766	0.821822	1.001782	1.101321	0.650871	0.38039	0.424885	0.645906	0.727167	0.707588	0.801491
16	C:\Users\Jake\Desktop\Image	0.696156	0.534875	0.857107	0.734074	0.500611	0.803504	0.786156	0.447578	0.475906	0.550821	0.730821

Figure 2.18: The Final Image Pairwise Distance Matrix

2.6.4. Hierarchical Agglomerative Clustering using R

While there are multitudes of packages and options for clustering data within R, the base language provides functions for simple HAC clustering [45]. The purpose of this section is not explain in too much detail how HAC clustering works. Rather, a demonstration of how HAC clustering can be used to identify similar images is provided.

The purpose of clustering is to divide a collection of items into groups based on their similarity to each other. The HAC clustering of images works by comparing the pairwise distances of all images and then grouping them into a structure called a dendrogram. The "dendrogram" is a map of all possible clustering assignments at various dissimilarity thresholds. The dissimilarity threshold dictates the maximum amount two images (or two clusters of images) are allowed to be dissimilar and still end up being merged into the same cluster.

Once the dendrogram has been created, all images can quickly be assigned to clusters using any dissimilarity threshold value, which is referred to as the "cut height". The cut height is typically provided by the user. This process can also occur in reverse with the user requesting a particular total number of clusters at which point the algorithm calculates the best cut height to achieve the requested result. While a

small cut height will produce smaller clusters with highly similar images, a large cut height will create larger clusters containing more dissimilar images.

The dendrogram maps out all possible clustering memberships based on each image's dissimilarity to the cluster as a group. This can be done using each cluster's minimum, maximum, average, or centroid distances. Depending on what measure is chosen, the clustering type is referred to as either single-linkage (minimum), complete-linkage (maximum), UPGMA (Unweighted Pair Group Method with Arithmetic Mean) (average), or centroid-based (centroid) clustering. While there are many types of clustering methods, these seem to be the most common.

The clustering algorithms typically work by starting out with all images in their own individual cluster of 1, and then successively combining the clusters which are in closest distance proximity based on the distance metrics described above. The clusters are combined until no more clusters can be joined without violating the provided cut height threshold. While this description is a slight over-simplification, additional research regarding this topic is left up to the reader. Figure 2.19 shows a dendrogram with all possible clustering combinations for 40 images at various cut heights which are displayed along the y axis.

Looking at the dendrogram in Figure 2.19, it is easy to see that a cut height of 0.60 would produce only two clusters containing all 40 images. In this case, the two clusters are very large and likely contain many dissimilar images since the cut height threshold allows images with a distance of up to 0.60 to be included within the same cluster. In the other extreme, a cut height of 0.10 places all but 2 images into singleton clusters containing only one image each. This is due to the fact that images must be at least 90% similar to be included within the same cluster. Using the demo application [50], the cut height can be adjusted to explore the impact on clustering similar images. Figure 2.20 illustrates how image clusters are changed by

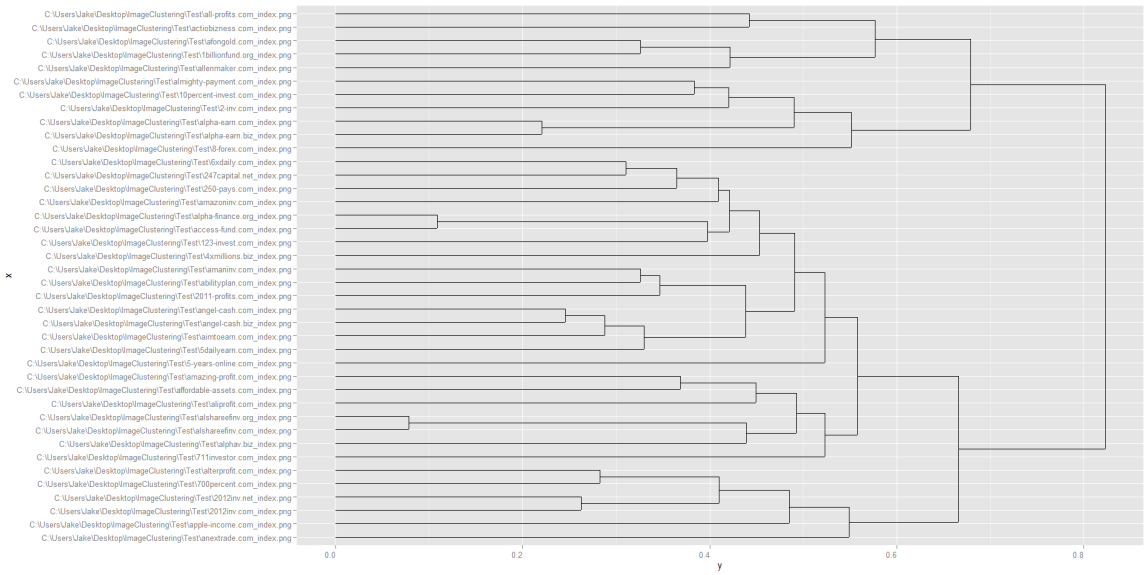


Figure 2.19: A Dendrogram for 40 Images

a 10% adjustment in cut height when using screen shots of the replicated criminal websites discussed in Chapter 6.

Figure 2.20: A 10% Decrease in Cut Height Removes the First Image from the Cluster

Figure 2.20 shows a 10% reduction in cut height forcing the third image out of the cluster. Since two of the images are highly similar, they remain in a cluster of 2 once the cut height dissimilarity threshold is reduced to 0.25. It is important to understand that the optimal cut height for image clustering will vary greatly depending on the types of images you are trying to cluster and the image features used to create the pairwise image distance matrix. Even within the demonstration project [50] sample images, strong arguments can made for adjustments in the cut height depending on

individual goals. For instance, Figure 2.21 shows 4 images included in a single cluster. However, some might argue that one of these images is sufficiently different than the other three to be excluded from the cluster.

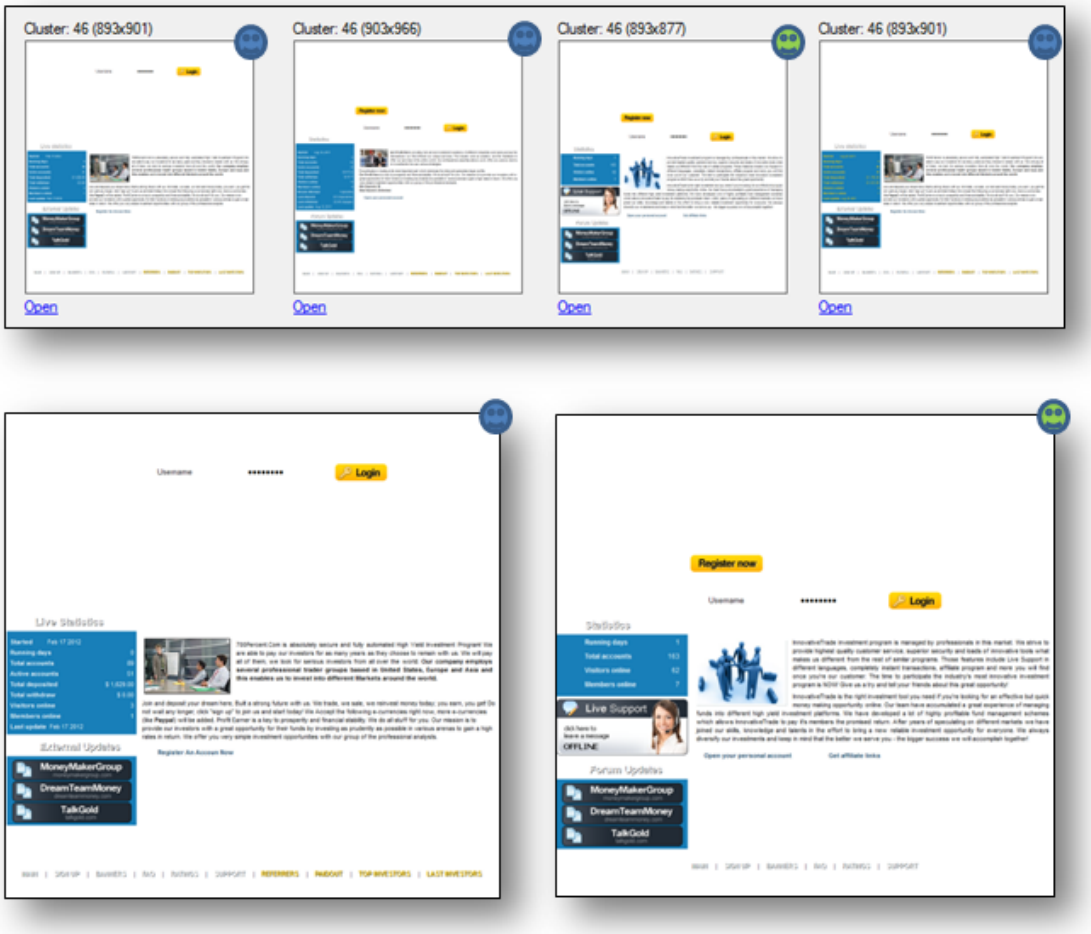


Figure 2.21: Should the Third Image be Excluded from the Cluster Since it Contains Different Graphics?

Conversely, if one were trying to identify websites made from the same template, all of the images above would be clustered acceptably. In fact, you would even want these images to be included in the same cluster, if they had different color schemes. In this instance, a more generous cut height might be applied, and in some cases, different features might be required for the image matching exercise at hand.

Once the distance matrix has been created, it is relatively straightforward to perform the clustering using R. The R program used to perform Hierarchical Agglomerative Clustering on the image distance matrix can be seen in Figure 2.22. The cut height dictates how strict the algorithm is on putting images in the same cluster. If the cut height (measure of dissimilarity) is set to .99 all images would be put in the same cluster. If the cut height is set to .1 almost all items would be placed in singleton clusters (clusters of size 1).

```
#read in the file and convert to a dist matrix, MUST!!! - convert to dist first before doing any subsetting or sorts
distMatrix<-read.csv(inputMatrixPath,sep=",",header=T,row.names=1)
#convert the input to a distance matrix of class "dist"
distMatrix<-as.dist(as.matrix(distMatrix[,1:length(distMatrix)]))

#cluster matrix data and create dendrogram
jclust<-hclust(distMatrix,method="average")

#use dynamic cut package to find the best cutHeight      below a certain threshold.
#cuts<-cutreeDynamic(jclust, cutHeight= cutHeight,
#        minClusterSize = 2, method = "tree", deepSplit = TRUE);

#Set your own cut height.
cuts<-cutree(jclust,h=cutHeight)

#Create the output file.
cutsOut <-data.frame(cbind(WebSite=jclust$labels,Cluster=cuts))

#convert the cluster number to int and sort it
cutsOut$Cluster <- as.integer(cutsOut$Cluster)
cutsOut<-cutsOut[ order(cutsOut[,2]), ]

# Write clustering output to a csv file.  Column names are removed since the data is read back into C#
write.table(cutsOut, file =clusteringOutput,row.names=FALSE, col.names=FALSE,sep=",",quote=FALSE)
```

Figure 2.22: HAC Clustering Using R

Once the image distance matrix is saved to disk using C#, the ImageClustering.r program reads in the file and converts it to an R distance matrix (dist) object. Next, the function hclust() creates the dendrogram mapping using the UPGMA (Unweighted Pair Group Method with Arithmetic Mean) or "average" distance method. The cutree() function then cuts the dendrogram to the "cutHeight" which is specified by the user. It is important to note that this value was actually specified by the user on the demo application's [50] form. This value can be passed as an argument from

C# to R using the Rscript.exe program which comes with the standard R installation. Rscript.exe can be started in a seperate process using C# and then passed any number of command line arguments which can be accessed and used within the R program.

2.6.5. Conclusion

The C# and R programming languages can be combined to create powerful parallel processing pipelines using MapReduce style programming and harnessing the analytical powers of R as needed. Information produced in both R and C# can also easily be exchanged and combined across programs when necessary. The demo application [50] provided uses the powerful parallel programming libraries in C# to rapidly extract and compare luminosity histograms from images creating a pairwise distance matrix for all images contained in a directory folder provided by the user. Next, C# uses R to perform HAC clustering to combine similar images and then display the output from R on the demo application's [50] form. The demo application [50] gives the user a thumbnail preview of the currently selected image row and also the next 3 images below the currently selected image. This allows the user to change the clustering cut height and quickly re-run the R clustering program until they are satisfied with the image clustering results.

Chapter 3

DEVELOPING FEATURES FOR BIOINFORMATICS

3.1. Introduction

Many popular gene sequence analysis tools are based on the idea of pairwise sequence alignment [116, 138]. Sequence alignment finds the least costly transformation of one sequence into another using insertions, deletions, and replacements. This method is related to the well know distance metric called Levenshtein or edit distance. However, finding the optimal pairwise alignment is computationally expensive and requires dynamic programming.

Gene sequence *words* are sub-sequences of a given length. In addition to words they are often also refered to as k-mers or n-grams, where k and n represent the word length. Words are extracted from individual gene sequences and used for similarity estimations between two or more gene sequences [146]. Methods like BLAST [11] were developed for searching large sequence databases. Such methods search for seed words first and then expand matches. These so-called alignment-free methods [146] are based on gene sequence word counts and have become increasingly popular since the computationally expensive sequence alignment method is avoided. One of the most successful word-based methods is the RDP classifier [151], a naive Bayesian classifier widely used for organism classification based on 16S rRNA gene sequence data.

3.2. Sequence Feature Extraction

Numerous methods for the extraction, retention, and matching of word collections from sequence data have been studied. Some of these methods include: 12-mer collections with the compression of 4 nucleotides per byte using byte-wise searching [11], sorting of k-mer collections for the optimized processing of shorter matches within similar sequences [55], modification of the edit distance calculation to include only removals (maximal matches) in order to perform distance calculations in linear time [143], and the use of locality sensitive hashing for inexact matching of longer k-mers [26].

This research combines the following two primary contributions in a novel and innovative way to achieve the results presented:

1. A form of locality sensitive hashing called *minhashing* is used to rapidly process much longer word lengths for enhanced accuracy. Minhashing allows us to estimate Jaccard similarity without computing and storing information for all possible words extracted from a gene sequence. Instead, we use the intersection of the minhash signatures produced during the minhashing process to quickly calculate an accurate approximation of the Jaccard similarity between sequences and known taxonomy categories.

2. A MapReduce style parallel pipeline is used to simultaneously identify unique gene sequence words, minhash each word generating minhash signatures, and intersect minhash signatures to estimate Jaccard similarity for highly accurate and efficient identification of gene sequence taxonomies.

Buhler [26] previously used locality sensitive hashing to allow for inexact matches between longer words of a predefined length. We use locality sensitive hashing in a very different way as a selection strategy for performing exact word matching when the number of possible words becomes much too large to store. For example, with an

alphabet of 4 symbols, the number of unique words of length 60 is 4^{60} which is already more than 10^{36} distinct words. The RDP classifier utilizes a fixed word length of only 8 bases to perform its taxonomy classification processing making the total possible number of unique words (i.e., features for the classifier) only $4^8 = 65,536$ words [151].

Strand is able to very rapidly classify sequences while still taking advantage of the increased accuracy provided by extracting longer words. Using the much larger possible feature space provided by a longer word length combined with locality sensitive hashing to reduce memory requirements, Strand achieves classification accuracy similar to RDP in processing times which are magnitudes of order faster. All stages of Strand processing are highly parallelized, concurrently mapping all identified words from a gene sequence and reducing mapped words into minhash signatures simultaneously. The unique relationship of Jaccard similarity between sets and locality sensitive hashing [129] allows minhashing to occur during learning, storing only a predetermined number of minimum hash values in place of all words extracted from the gene sequences in the learning set. This process reduces the amount of memory used during learning and classification to a manageable amount.

3.3. Background

This research combines the three concepts of longer length word extraction, minhashing, and multicore / multimachine MapReduce style processing in a novel way that produces superior gene sequence feature extraction and classification results. The following sections briefly describe the background material relevant to this research.

3.3.1. Word Extraction

The general concept of k-mers or words was originally defined as n-grams during 1948 in an information theoretic context [136] as a subsequence of n consecutive

Figure 3.1: Gene Sequence Words of Length 8 Extracted using a Sliding Window

symbols. We will use the terms *words* or k-mers to refer to n-grams created from a gene sequence. Over the past twenty years, numerous methods utilizing words for gene sequence comparison and classification have been presented [146]. These methods are typically much faster than alignment-based methods and are often called alignment-free methods. The most common method for word extraction uses a sliding window of a fixed size. Once the word length k is defined, the sliding window moves from left to right across the gene sequence data producing each word by capturing k consecutive bases from the sequence. Figure 3.1 illustrates how gene sequence words or subsequences of length 8.

In Chapter 4, Strand performs word extraction using lock-free data structures to identify unique gene sequence words. This method is similar to other highly parallel word counting tools such as Jellyfish [103]. Traditionally, computationally expensive lock objects are used in parallel programs to synchronize thread-level access to a shared resource. Each thread must either acquire the lock object or block until it becomes available prior to entering critical sections of code. Lock-free data structures avoid the overhead associated with locking by making use of low-level atomic read/write transactions and other lock-free programming techniques.

3.3.2. Minhashing

In word-based sequence comparison, sequences are often considered to be sets of words. A form of locality sensitive hashing called minhashing uses a family of random hash functions to generate a minhash signature for each set. Each hash function used in the family of n hash functions implement a unique permutation function, imposing an order on the set to be minhashed. Choosing the element with the minimal hash value from each of the n hash functions results in a signature of n elements.

Typically the original set is several magnitutes larger than n resulting in a significant reduction of the memory required for storage. From these signatures an estimate of the Jaccard similarity between two sets can be calculated [23, 129]. This means that when a minhash function is used to randomly select values from a from a set, the proportion of matching values drawn is an accurate approximation of the Jaccard similarity between the two sets [93]. This approximation of Jaccard similarity increases in accuracy as the minhash signature sample size increases [93].

Minhashing has been successfully applied in numerous applications including estimating similarity between images [24] and documents [23], document clustering on the internet [25], image retrieval [32], detecting video copies [29], and relevant news recommendations [98].

In this research, we apply minhashing to estimate the similarity between sequences which have been transformed into very large sets of words.

3.3.3. MapReduce Style Processing

MapReduce style programs break algorithms down into *map* and *reduce* steps which represent independent units of work that can be executed using parallel processing [31, 48]. Initially, input data is split into many pieces and provided to multiple instances of the mapping functions executing in parallel. The result of mapping is a

key-value pair including an aggregation key and its associated value or values. The key-value pairs are redistributed using the aggregation key and then processed in parallel by multiple instances of the reduce function producing an intermediary or final result.

MapReduce is highly scalable and has been used by large companies such as Google and Yahoo! to successfully manage rapid growth and extremely massive data processing tasks [106]. Over the past few years, MapReduce processing has been proposed for use in many areas including: analyzing gene sequencing data [106], machine learning on multiple cores [84], and highly fault tolerant data processing systems [31, 40].

In Chapter 4, Strand uses MapReduce style processing to quickly map gene sequence data into words while simultaneously reducing the mapped words from multiple sequences into their appropriate corresponding minhash signatures.

3.4. Extracting Bioinformatics Features for Abundance Estimation

Modern sequencing technologies are continually producing increasing volumes of sequence data used in the many fields of computational molecular biology. Classification of DNA sequences is still challenging due to the growing number of new species which have been sequenced over the last decade and the sheer volume of sequence data produced. Metagenomics focuses on the study of sequences extracted directly from a given sample, and researchers continue to generate and classify sequences from many diverse environments. In 1991, the Human Genome Project performed DNA sequencing on 600 randomly selected human brain complementary DNA (cDNA) clones finding over 337 new genes with strong similarities to 48 genes from other organisms [8]. During 2007, the Human Microbiome Project acknowledged humans as "supraorganisms" comprising both human and microbial components and set out to

better understand the microbial components which comprise the human genetic and metabolic landscape [141].

Samples of 16S rRNA have been used to analyze the microbial communities that exist on the forearm [59], vagina [71], colon [54], stomach [20], and esophagus [124]. It is becoming increasingly common for research to connect specific afflictions to distinct microbial compositions within the human body. For example, the identification of decreased Bacteroidetes microbes within the gut has been linked to obesity [97]. Research [96] has also shown that obesity alters gut microbial ecology. Feces is a common sampling source for distinguishing differences in the gut microbial diversity and composition of humans [96]. Shotgun sequencing typically analyzes sequence data which is representative of multiple species and attempts to describe the abundance of the various organisms identified within the sample. Sequences from many other environments outside the human body have been studied including diverse sources such as bread [22], pigs [154], acidic mine drainage [142], and saltwater [145]. Venter et. al. studied sequenced saltwater samples from the Sargasso Sea, and shotgun sequencing revealed that the 1.045 billion base pairs sequenced were estimated to derive from at least 1800 genomic species [145].

The individual metagenomics reads generated by common sequencing tools are relatively short in length. Publically available sample data sequenced by the popular Illumina HiSeq and MiSeq sequencing platforms had a mean length of 92 and 156 base pairs per read respectively [157]. For a given sample, millions of reads may be produced. During abundance estimation, the goal is to identify as many microbial species as possible to determine the sample's microbial composition. This is accomplished using reads created from the sample provided for sequencing. While it is possible to use the sequence alignment tool BLAST [12] for comparing sample sequences to other sequences from known taxonomies, many newer sequence classification tools claim to

be faster and/or more accurate [52, 56, 85, 121, 151, 157].

3.4.1. Word Extraction for Abundance Estimation

Words are extracted from individual gene sequences and used for similarity estimations between two or more gene sequences [146]. Methods like BLAST [12] were developed for searching large sequence databases. Such methods search for seed words first and then expand matches. These so called alignment-free methods [146] are based on gene sequence word counts and have become increasingly popular since the computationally expensive sequence alignment method is avoided. The most common method for word extraction uses a sliding window of a fixed size. Once the word length k is defined, the sliding window moves from left to right across the gene sequence data producing each word by capturing k consecutive bases from the sequence.

Rapid abundance estimation tools [52, 121, 157], including Strand, derive a large speed advantage by utilizing an exact-match between the words extracted from sequence data to identify the similarity between two sequences. However, this approach comes at the cost of storing a very large number of sequence words to make accurate classifications when the value for k results in a very long word. For example, the extraction of $k = 30$ base words results in $4^{30} \approx 10^{18}$ unique word possibilities within the training data feature space when an alphabet of 4 symbols (A, C, G, T) is considered. Other sequence classifiers avoid storing large volumes of words by reducing the value for k and the total possible feature space size for the training data structure.

The RDP classier [151] utilizes a fixed word length of only 8 bases to perform its taxonomy classification processing making the total possible number of unique words (i.e., features for the classier) only $4^8 = 65,536$ words. Unfortunately, such a small feature space makes distinguishing between many sequence classes challenging as the probability for finding duplicate sequence words greatly increases when compared to

the longer 30 base word length. In contrast, Strand avoids these training data feature space challenges by utilizing a form of lossy compression called Minhashing which is discussed later in Section 3.4.3. The Minhashing process converts sequence words into a 64-bit integer making the total Strand training data feature space size 2^{64} possible unique words.

3.4.2. Creating Words from Sequences that do not Fit into Memory

Working with large amounts of unstructured data (e.g., gene sequences) has become important for many business, engineering, and scientific applications. Locality sensitive hashing systems drastically reduce the time required to perform a similarity search in high dimensional space (e.g., created by the words in the vector space model for gene sequences). Locality sensitive hashing also dramatically reduces the amount of data required for storage and comparison by applying probabilistic dimensionality reduction. We concentrate on the implementation of min-wise independent permutations (*minhashing*) which provides an efficient way to determine an accurate approximation of the Jaccard similarity coefficient between sets (e.g., sets of terms in documents or sets of words extracted from gene sequences) [63, 76].

While longer gene sequence words produce more accurate exact match Jaccard coefficient approximations (See Chapter 4 for detailed analysis), these words also take up more space in memory. For example, a single C# string is estimated to take approximately 20 bytes of object data overhead and an additional 2 bytes per character of required memory space [125]. This is approximately 82 bytes or 656 bits of space required to store a 31 base string in memory. While hashing words into 32 or 64-bit integers can reduce the memory footprint by very large amounts, the selected word length can also influence collisions, drastically impacting classification accuracy.

Figure 3.2 illustrates how the available unique hash values per unique gene sequence word and potential collisions produced are influenced by both word length

31 Base Words Into 32 Bit Hash Codes	
Hash: 2^32 (32 Bit Int)	4,294,967,296
Sequence: 4^31 (31 Base Word)	4,611,686,018,427,390,000
Avg. Sequence Words Per Hash Value	1,073,741,824
31 Base Words Into 64 Bit Hash Codes	
Hash: 2^64 (64 Bit Int)	18,446,744,073,709,600,000
Sequence: 4^31 (31 Base Word)	4,611,686,018,427,390,000
Avg. Sequence Words Per Hash Value	0.25
16 Base Words Into 32 Bit Hash Codes	
Hash: 2^32 (32 Bit Int)	4,294,967,296
Sequence: 4^16 (16 Base Word)	4,294,967,296
Avg. Sequence Words Per Hash Value	1
32 Base Words Into 64 Bit Hash Codes	
Hash: 2^64 (64 Bit Int)	18,446,744,073,709,600,000
Sequence: 4^32 (32 Base Word)	18,446,744,073,709,600,000
Avg. Sequence Words Per Hash Value	1

Figure 3.2: Average Collisions Per Gene Sequence Word using 16 and 32 Base Words and 32 vs. 64-Bit Hash Codes

and the selection of an appropriate hash code size. For example, over 1 billion potential collisions per word are observed when hashing a 31 base gene sequence word into only 32-bits while a 64-bit hash provides ample room for each unique 31 base word value. Finally, the remaining sections of Figure 3.2 show that a 32-bit hashing function provides enough unique values to map words up to 16 bases in length, while a 64-bit hashing function supports unique values for words up to 32 bases in length. Collisions may still occur when storing a 31 base word as a 64-bit hash. However, they are drastically reduced when compared to storing a 31 base word as a 32-bit hash.

3.4.3. Creating Minhash Signatures from Sequences that do not Fit into Memory

The concept of locality sensitive hashing is well established with publications dating back as far as 1998 [74] and 1999 [63] exploring its use for breaking the curse of

dimensionality in nearest neighbor query problems. Prior to locality sensitive hashing, the data structures used for similarity searches scaled very poorly. Without using a method for approximation such as locality sensitive hashing, searches exceeding 10 to 20 dimensions, required inspection of most records in the database similar to a brute force linear search.

In word-based sequence comparison, sequences are often considered to be sets of words. A form of locality sensitive hashing called minhashing uses one or more random hash functions to generate a minhash signature for each set. Each hash function used implements a unique permutation function, imposing an order on the set to be minhashed. Choosing one or more elements with the minimal hash value from each of the n permutations of the set results in a signature of n elements. Typically the original set is several magnitutes larger than n resulting in a significant reduction of the memory required for storage. From these signatures an estimate of the Jaccard similarity between two sets can be calculated [23, 93]. Minhashing has been successfully applied in numerous applications including estimating similarity between images [24] and documents [23], document clustering on the internet [25], image retrieval [32], detecting video copies [29], and relevant news recommendations [98].

When using only a single hashing function, hash values are sorted, and the smallest n values are selected to form the minhash signature. This approach is advantageous when very large volumes of sequence data must be processed. Using a single hashing function avoids the overhead required for each word to be hashed by multiple hashing functions. This approach is shown in Chapter 4 to have advantages over merely taking random samples when comparing sequence data. Since the hashing function imposes a random permutation on the gene sequence words, selecting the n smallest words from any number of hashing functions into a minhash signature ensures that matching words are efficiently extracted from both sequences when they exist.

Some sequenced data contains millions of bases which may not fit into memory during word extraction. In these instances, it may not be possible to generate the minhash hash signature for a particular sequence or human genome all at one time. A process called "chunking" can be applied to break sequences in to smaller pieces for creating the minhash signature in multiple stages. Chunking has two primary advantages.

1. Sequences can be "chunked" to select minimum hash values from portions of the sequence in stages. This allows only portions very large sequences to be "buffered" into memory in stages.

2. Chunking may also be deployed to enforce and guarantee the selection of minhash values evenly across the entire sequence. Using this strategy, a much smaller number of minhash values are selected from each chunk of a given sequence and the user is ensured that an even number of minhash values are generated from each selected section or chunk.

In a typical chunking process, overlapping chunks of text are selected from the target sequence. The chunk overlap ensures that no words within the entire gene sequence are broken by the chunking process. When producing 31 base words with a chunk size of 100 words for example, bases 1 - 130 are selected for chunk 1 and bases 101 - 230 selected for chunk 2. This ensures that exactly 100 words of 31 bases in length are produced from both chunk 1 and chunk 2 with no loss of gene sequence words between the two chunks. In this case, a chunk 1 and 2 overlap of 30 words occurs between positions 101 - 130 to ensure that the 100th word in chunk one is exactly 31 bases in length and that the 1 word in chunk 2 starts exactly 1 base after the last word in chunk 1.

3.5. Conclusion

Each of the feature extraction methods for gene sequences introduced in this Chapter support various aspects of the rapid and accurate sequence classification techniques demonstrated by the Strand application in Chapter 4. In the next Chapter, the Strand application is utilized on various sequence datasets and compared to other state of the art gene sequence classification tools demonstrating how the previously described sequence feature extraction techniques may be used to produce superior gene sequence classification and abundance estimation results.

Chapter 4

S.T.R.A.N.D.

The Super Threaded Reference-Free Alignment-Free N-sequence Decoder (Strand) is a highly parallel technique for the learning and classification of gene sequence data into any number of associated categories or gene sequence taxonomies. Current methods, including the state-of-the-art sequence classification method RDP, balance performance by using a shorter word length. Strand in contrast uses a much longer word length, and does so efficiently by implementing a Divide and Conquer algorithm leveraging MapReduce style processing and locality sensitive hashing. Strand is able to learn gene sequence taxonomies and classify new sequences approximately 20 times faster than the RDP classifier while still achieving comparable accuracy results. This paper compares the accuracy and performance characteristics of Strand against RDP using 16S rRNA sequence data from the RDP training dataset and the Greengenes sequence repository.

This research combines the following two primary contributions in a novel and innovative way to achieve the results presented:

1. A form of locality sensitive hashing called *minhashing* is used to rapidly process much longer word lengths for enhanced accuracy. Minhashing allows us to estimate Jaccard similarity without computing and storing information for all possible words extracted from a gene sequence. Instead, we use the intersection of the minhash signatures produced during the minhashing process to quickly calculate an accurate approximation of the Jaccard similarity between sequences and known taxonomy categories.

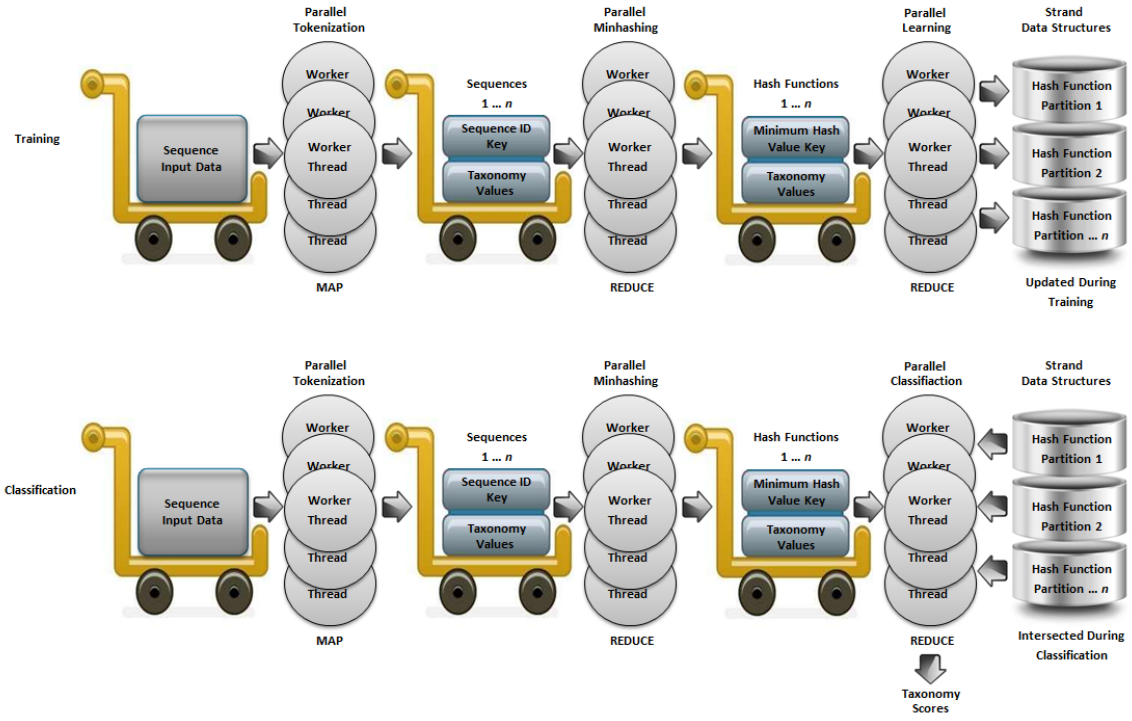


Figure 4.1: High-level Diagram of the Strand MapReduce Style Pipeline.

2. A MapReduce style parallel pipeline is used to simultaneously identify unique gene sequence words, minhash each word generating minhash signatures, and intersect minhash signatures to estimate Jaccard similarity for highly accurate and efficient identification of gene sequence taxonomies.

4.1. Learning Category Signatures

Figure 4.1 illustrates a high-level process of the Strand MapReduce pipeline including both the mapping of gene sequence data into words, the reduction of words into minimum hash values, and finally, the last reduce step which organizes the minhash signatures by category. In the following we will describe each stage in detail.

4.1.1. Mapping Sequences into Words

The input data are sequences with associated categories.

Definition 4.1 [Sequence] Let S be a single input sequence, a sequence of $|S|$ symbols from alphabet $\Sigma = \{A, C, G, T\}$.

Definition 4.2 [Category] Let C be the set of all L known taxonomic categories and $c_l \in C$ be a single category where $l = \{1, 2, \ldots, L\}$. Each sequence S is assigned a unique true category $c_l \in C$.

The goal of mapping sequences into words is to create for each sequence a word profile.

Definition 4.3 [Sequence Word Profile] Let $\mathcal{S}$ denote the word profile of sequence S, i.e., the set of all words $s_j \in \mathcal{S}, j = \{1, 2, \ldots, |\mathcal{S}|\}$, with length k extracted from sequence S.

4.1.2. Creating Sequence Minhash Signatures

As words are produced, minhashing operations are also performed simultaneously in parallel to create minhash signatures.

Definition 4.4 [Sequence Minhash Signature] Minhashing (min-wise locality sensitive hashing) applies a family of random hashing functions $h_1, h_2, h_3 ... h_k$ to the input sequence word profile $\mathcal{S}$ to produce k independent random permutations and then chooses the element with the minimal hash value for each. We define the minhash function:

$$\text{minhash: } S \rightarrow \mathbb{Z}_+^k$$

which maps a sequence word profile $\mathcal{S}$ to a set of k minhash values $\mathcal{M} = \{m_1, m_2, \ldots, m_k\}$, called the minhash signature.

Min-wise locality sensitive hashing operations are performed in parallel and continually consume all randomly selected word hashes for each processed sequence. The minhash signature's length is predetermined by the number of random hashing functions used during minhash processing, and the minhash signature length impacts processing time and overall classification accuracy. Processes using more hashing functions (i.e., longer minhash signatures) have been proven to produce more accurate Jaccard estimations [129]. However, careful consideration must be given to the trade-off between the minhash signature's impact on performance and Jaccard estimation accuracy.

A thread-safe minhash signature collection contains one minhash signature for each unique input sequence. During minhash processing, all hash values produced for each sequence's unique set of word hashes are compared to the sequence's current minhash signature values. The minimum hash values across all unique words for each sequence and each unique hashing function are then retained within each sequence's final minhash signature. In applications where the similarity calculation incorporates word length or the frequency of each minimum hash value, the length and frequency for any word resulting in a particular minimum hash value can also be contained as additional properties within each minhash signature's values. However, lengths are not retained within the Strand data structure during training since they can quickly be determined during any subsequent classification's minhashing process. In some cases, the total number of hashing functions and overall minhashing operations can be reduced by saving the n smallest hash values generated from each individual hashing function. For instance, the number of hashing operations can be cut in half by retaining the 2 smallest values produced from each unique hashing function.

4.1.3. Reducing Sequence Minhash Signatures into Category Signatures

Next we discuss how to represent an entire category as a signature built from the minhash signatures of all sequences in the category.

Definition 4.5 [Category Minhash Signature] We define the category minhash signature of category $c_l \in \mathcal{C}$ as the union of the sequence minhash signatures of all sequences assigned to category cl:

$$\mathcal{C}_l = \bigcup_{\mathcal{S} \in c_l} \text{minhash}(S),$$

where the union is calculated for each minhash hashing function separately.

The Strand data structure actually represents an array containing one data structure for each unique hashing function used (see Figure 4.1). Since this structure is keyed using minhash values, hash function partitions must exist to separate the minhash values produced by each unique hashing function. Within each individual hash function partition, a collection of key-value pairs (kvp) exists which contains the minhash value as a key and then a second nested collection of categorical key-value pairs for each value. The nested collection of kvp-values contains all category numbers and associated frequencies (when required) that have been encountered for a particular minhash value. In practice however, minhash values seldom appear to be associated with more than one taxonomy category which drastically reduces the opportunity for imbalance between categories, especially when minhash value frequencies are not used within the classification similarity function.

During learning, minimum hash values for each unique hashing function are retained within the array of nested categorical key-value pair collections and partitioned by each unique hashing function. Each hash function's collection contains all unique minimum hash values, their associated taxonomies, and optional frequencies. Us-

ing the Strand data structure, minhash signatures for each input data sequence can quickly be compared to all minimum hash values associated with each known taxonomy including the taxonomy frequency (when utilized) during classification. During training, each value in the input data sequence's minhash signature is reduced into the Strand data structure by either creating a new entry or adding additional taxonomy categories to an existing entry's nested categorical key-value pair collection.

All results presented in this research were achieved using only a binary classification similarity function. This approach produced optimal performance while still achieving comparable accuracy results when benchmarked against the current top performer in this domain.

4.2. Classification Process

The MapReduce style architecture used for learning and classification are very similar. While the process of mapping words into minhash signatures is identical, the reduce function now instead of creating category signatures creates classification scores.

The word profiles of sequences are the set of words contained within the sequences. A common way to calculate the similarity between sets is the Jaccard index. The Jaccard index between two sequence word profiles $\mathcal{S}_1$ and $\mathcal{S}_2$ is defined as:

$$\text{Jaccard}(\mathcal{S}_1, \mathcal{S}_2) = \frac{|\mathcal{S}_1 \cap \mathcal{S}_2|}{|\mathcal{S}_1 \cup \mathcal{S}_2|}$$

However, after minhashing we only have the sequence minhash signatures

$\mathcal{M}_1 = \text{minhash}(\mathcal{S}_1)$ and $\mathcal{M}_2 = \text{minhash}(\mathcal{S}_2)$

representing the two sequences. Fortunately, minhashing [129] allows us to efficiently estimate the Jaccard index using only the minhash signatures:

$$\text{Jaccard}(\mathcal{S}_1, \mathcal{S}_2) \approx \frac{|minhash(\mathcal{S}_1) \cap minhash(\mathcal{S}_2)|}{k},$$

where the intersection is taken hash-wise, i.e., how many minhash values agree between the two signatures.

Next, we discuss scoring the similarity between a sequence minhash signature and the category minhash signatures used for classification. Category signatures are not restricted to k values since they are created using the unique minhash values of all sequence minhash signatures belonging to the category. This is why we do not directly estimate the Jaccard index, but define a similarity measure based on the number of collisions between the minhash values in the sequence signature and the category signature.

Definition 4.6 [Minhash Category Collision] We define the Minhash Category Collision between a sequence S represented by the minhash signature $\mathcal{M}$ and a category signature $\mathcal{C}$ as:

$$\text{MCC}(\mathcal{M}, \mathcal{C}) = |\mathcal{M} \cap \mathcal{C}|,$$

where the intersection is calculated for each minhash hashing function separately.

We calculate MCC for each category and classify the sequence to the category resulting in the largest category collision count.

Many other more sophisticated approaches to score sequences are possible. These are left for future research.

4.3. Results

In this section we report on a set of initial experiments. First, we compare different word sizes and numbers of sequence signature lengths (i.e., the number of hashing functions used for minhashing). Then we compare Strand with RDP using two different data sets.

The data sets we use are all 16S rRNA data sets. The RDP classifier raw training set was obtained from the RDP download page[1]. It contains 9,217 sequences. The second data set we use is extracted from the Greengenes database[2]. We randomly selected 150,000 unaligned sequences with complete taxonomic information for our experiments. We used for all experiments 10-fold cross-validation. During 10-fold cross-validation, the entire training file is randomly shuffled and then divided into ten equally sized folds or segments. While nine folds are learned, one fold is held out for classification testing. This process is repeated until all ten folds have been held out and classified against.

The experiments were performed on a Windows 8 (64-bit) machine with a 2.7 Ghz Intel i7-4800MQ quad core CPU and 28 GB of main memory installed. For the RDP classifier we used version 2.5, and we implemented Strand in C#.

4.3.1. Choosing Word Size and Signature Length

Both, the used word size and the length of the signature need to be specified for Strand. We expect both parameters to have an impact on classification accuracy, space, and run time. While it is clear that with increasing signature lengths also the time needed to compute sequence signatures and the space needed to store signatures increases, the impact of word size is not so clear. In the following we will empirically find good values for both parameters. To look at the impact of the word length,

[1] http://sourceforge.net/projects/rdp-classifier/
[2] http://greengenes.lbl.gov

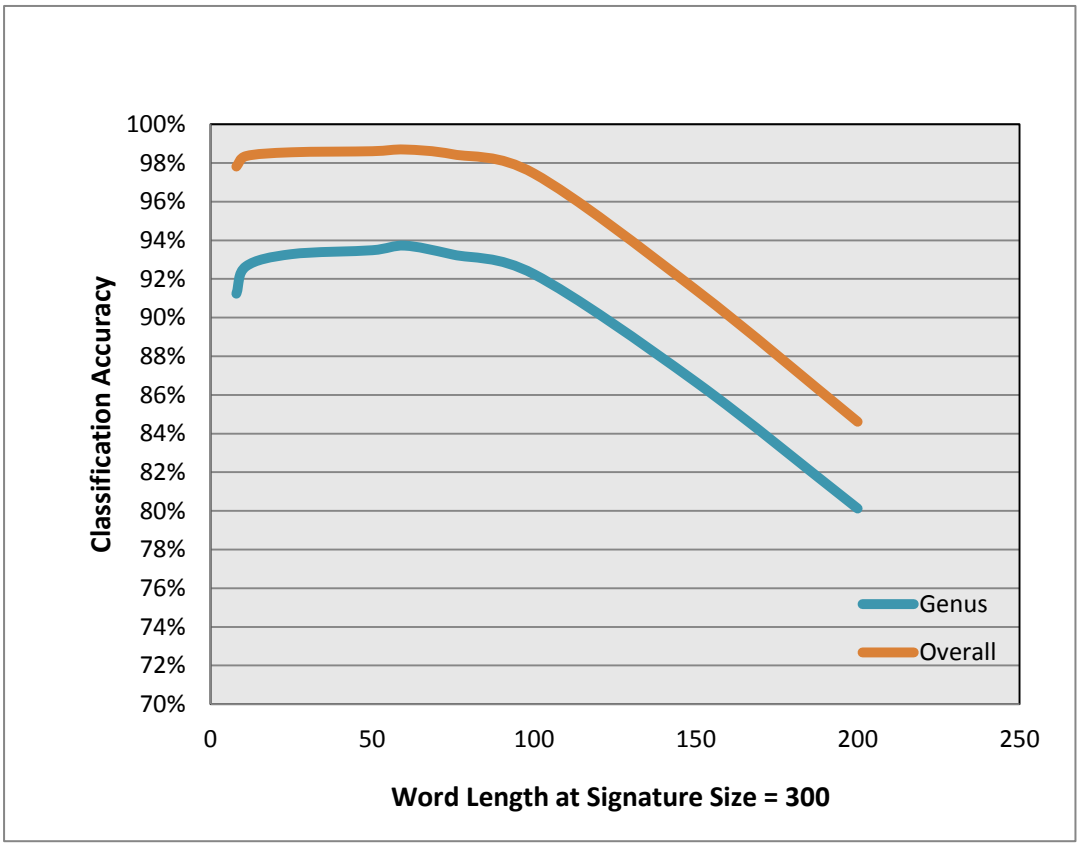


Figure 4.2: Strand word size accuracy on RDP 16S rRNA.

we set the signature size (i.e., number of hash functions used for minhashing) to 300. This was empirically found to be a reasonable value. Next we perform 10-fold cross-validation on the RDP training data for different word lengths ranging from 8 bases to 200 bases. The average accuracy of Genus prediction and overall prediction (average accuracy over all phylogenetic ranks) depending on the word length is shown in Figure 4.2. We see that accuracy increases with word length till the word length reaches 60 bases and then starts to fall quickly at lengths larger than 100 bases. This shows that the optimal word length for the used 16S rRNA is around 60 bases.

Next, we look at the impact of sequence signature length. We use a fixed word

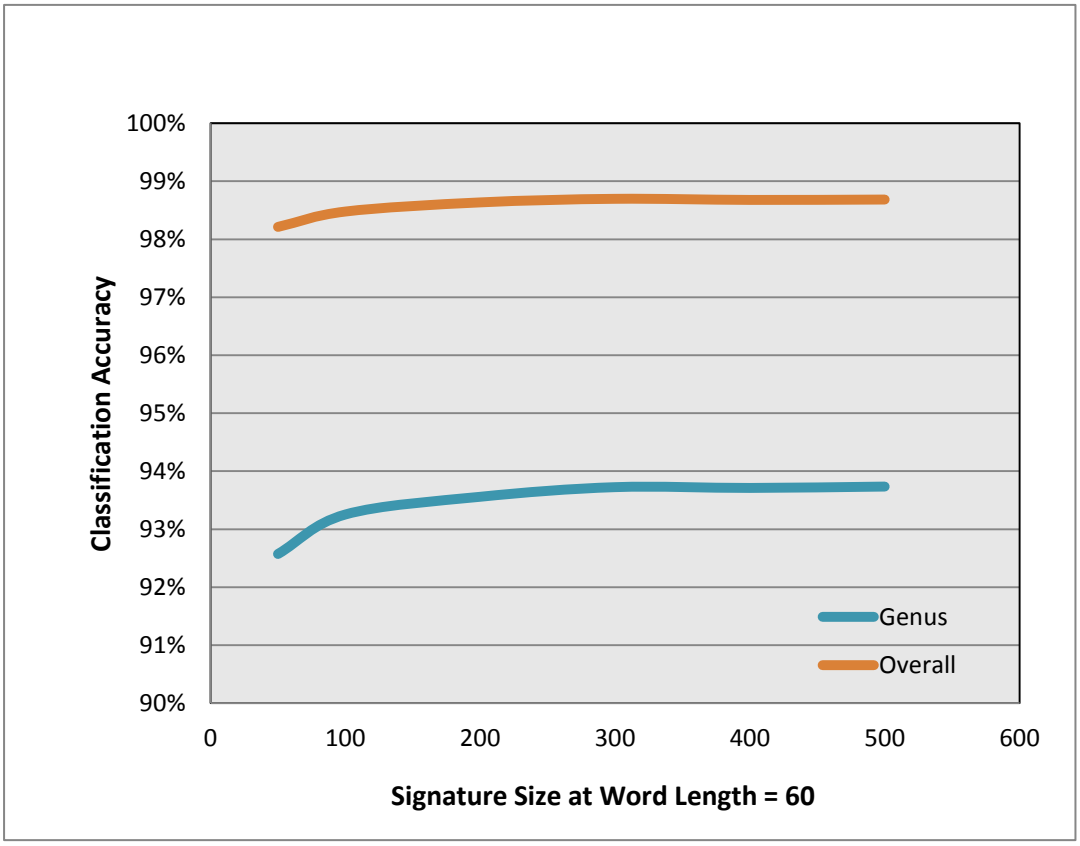


Figure 4.3: Strand signature length accuracy on RDP 16S rRNA.

size of 60 and perform again 10-fold cross-validation on the RDP training data set using signature lengths from 50 to 500. Figure 4.3 shows the impact of signature length. Accuracy initially increases with signature length, but flattens at about 300. Since an increased signature length directly increases run time (more hashes need to be calculated) and storage, we conclude that 300 is the optimal size for the used 16S rRNA data, but signature lengths of 200 or even 100 also provide good accuracy at lower computational cost.

While Strand allows users to specify word and signature sizes, empirically finding good values for both parameters need not be performed with each use of the applica-

tion. We believe the results presented show that using the Strand default word size of 60 bases and a signature size of 300 will produce optimal results on 16S rRNA sequence data.

4.3.2. Comparison of Strand and RDP on the RDP Training Data

In this section we compare Strand and RDP in terms of run time and accuracy. For the comparison we use again 10-fold cross-validation on the RDP training data set. Table 4.1 shows the run time for learning and classification using Strand and RDP. While the learning times for Strand are approximately 30% faster, Strand classifies sequences almost 20 times faster than RDP. Strand trained with 8,296 sequences averaging around 23 seconds per fold while RDP averaged around 33 seconds on the same 8,296 training sequences. During 10-fold cross-validation, Strand was able to classify 921 sequences averaging 3 seconds per fold while RDP's average classification time was 59 seconds per fold. This is substantial since classification occurs much more frequently than training in a typical application. Since Strand uses longer words during training and classification, no bootstrap sampling or consensus is used to determine taxonomy assignments. Strand greatly increases classification speeds when compared to RDP by combining this factor with a highly parallel MapReduce style processing pipeline.

An accuracy comparison between Strand and RDP is shown in Table 4.2. In cross-validation it is possible that we have a sequence with a Genus in the test set for which we have no sequence in the learning set. We exclude such sequences from the results since we cannot predict a Genus which we have not encountered during learning. Strand achieves similar overall accuracy to RDP, however, as we saw above in a fraction of the time.

Fold	Learning Time	Sequences Learned	Classification Time	Sequences Classified
		Strand Performance Results		
1	0:19	8,296	0:03	921
2	0:22	8,296	0:03	921
3	0:22	8,296	0:03	921
4	0:23	8,296	0:03	921
5	0:24	8,296	0:03	921
6	0:25	8,296	0:03	921
7	0:24	8,296	0:04	921
8	0:25	8,296	0:03	921
9	0:24	8,296	0:03	921
10	0:23	8,296	0:03	921
Avg.	**0:23**		**0:03**	
		RDP Performance Results		
1	0:33	8,296	0:58	921
2	0:33	8,296	0:58	921
3	0:33	8,296	0:59	921
4	0:33	8,296	0:59	921
5	0:34	8,296	1:00	921
6	0:34	8,296	0:59	921
7	0:33	8,296	0:59	921
8	0:33	8,296	0:58	921
9	0:32	8,296	0:57	921
10	0:33	8,296	0:58	921
Avg.	**0:33**		**0:59**	

Table 4.1: 10-fold cross-validation performance comparison between Strand and RDP.

Fold	Kingdom	Phylum	Class	Order	Family	Genus	Overall
Strand Accuracy Results							
1	100%	100%	99.9%	99.5%	99.0%	94.5%	98.8%
2	100%	100%	100%	100%	99.4%	95.2%	99.1%
3	100%	100%	99.8%	99.4%	98.3%	93.7%	98.5%
4	100%	100%	99.6%	99.3%	97.9%	93.1%	98.3%
5	99.9%	99.9%	99.9%	99.8%	99.4%	94.4%	98.9%
6	100%	100%	99.8%	99.5%	98.5%	93.7%	98.6%
7	100%	100%	100%	99.6%	99.2%	93.7%	98.8%
8	100%	100%	100%	99.9%	98.2%	92.5%	98.5%
9	100%	100%	100%	100%	99.4%	93.1%	98.8%
10	100%	100%	99.9%	99.8%	98.8%	93.3%	98.7%
Avg.	100%	100%	99.9%	99.7%	98.8%	93.7%	**98.7%**
RDP Accuracy Results							
1	100%	100%	100%	99.5%	98.6%	95.1%	98.9%
2	100%	100%	100%	99.8%	98.8%	94.0%	98.8%
3	100%	99.9%	99.8%	98.9%	97.6%	93.3%	98.3%
4	100%	100%	99.8%	99.5%	99.1%	93.2%	98.6%
5	100%	100%	100%	99.9%	99.5%	94.4%	99.0%
6	100%	100%	99.9%	99.5%	98.2%	92.3%	98.3%
7	100%	100%	100%	99.5%	99.2%	93.9%	98.8%
8	100%	100%	100%	99.5%	98.2%	91.5%	98.2%
9	100%	99.6%	99.6%	99.6%	99.0%	93.9%	98.6%
10	100%	100%	99.8%	99.5%	98.6%	92.3%	98.4%
Avg.	100%	100%	99.9%	99.5%	98.7%	93.4%	**98.6%**

Table 4.2: 10-fold cross-validation accuracy comparison between Strand and RDP.

4.3.3. Comparison of Strand and RDP on the Greengenes Data

Here we compare Strand and RDP on a sample of 150,000 sequences from the Greengenes project. While the RDP training set is relatively small and well curated to create a good classifier, these sequences will contain more variation. To analyze the impact of data set size, we hold 25,000 sequences back for testing and then use incrementally increased training set sizes from 25,000 to 125,000 in increments of 25,000 sequences. For Strand we use a word size of 60 and a sequence signature length of 100.

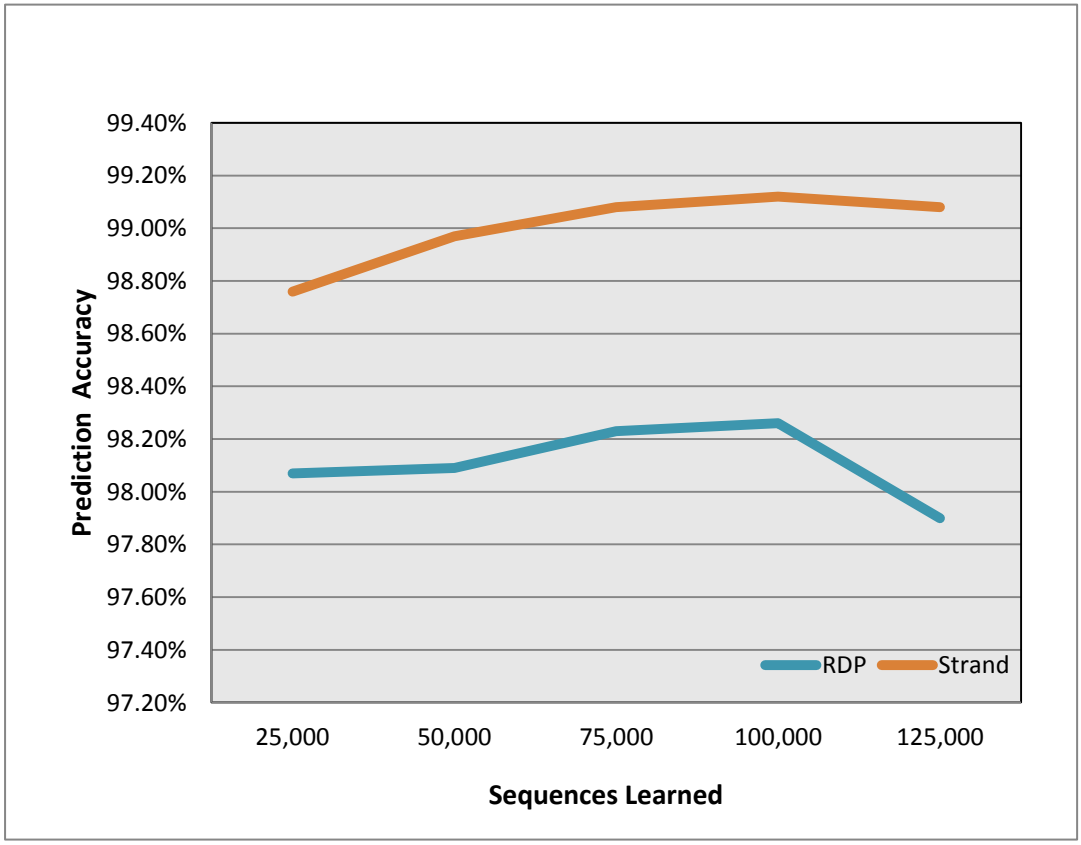


Figure 4.4: Strand and RDP accuracy on Greengenes data.

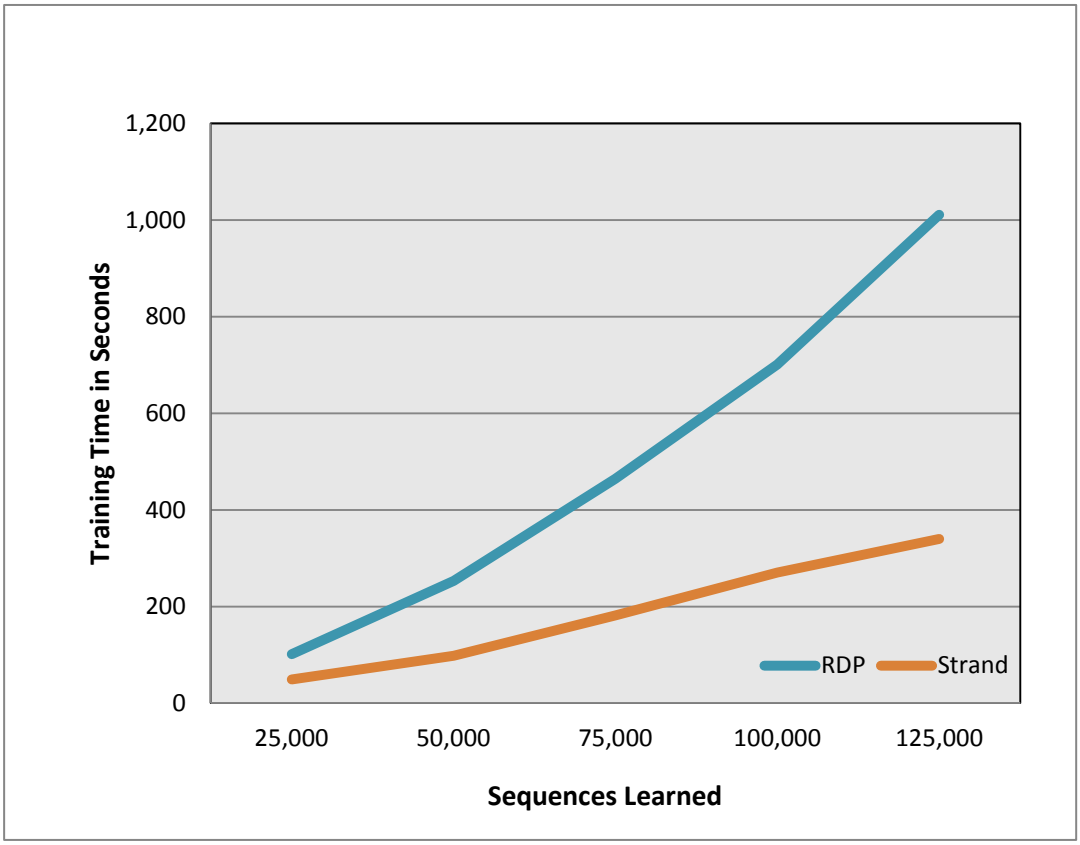


Figure 4.5: Strand and RDP running time on Greengenes data.

Figure 4.4 shows the classification accuracy of Strand and RDP using an increasingly larger training set. Strand has slightly higher accuracy. Accuracy increases for both classifiers with training set size. However, it is interesting that after 100,000 sequences, the accuracy starts to drop with a significantly steeper drop for RDP.

Figure 4.5 compares the time needed to train the classifiers with increasing training set size. During training, Strand execution times consistently outperform RDP with training time deltas further widening as input training volumes increase. In Figure 4.5 RDP training times increase rapidly as the training set size increases. Strand training times increase at a rate closer to linear. When training against 125,000 Greengenes

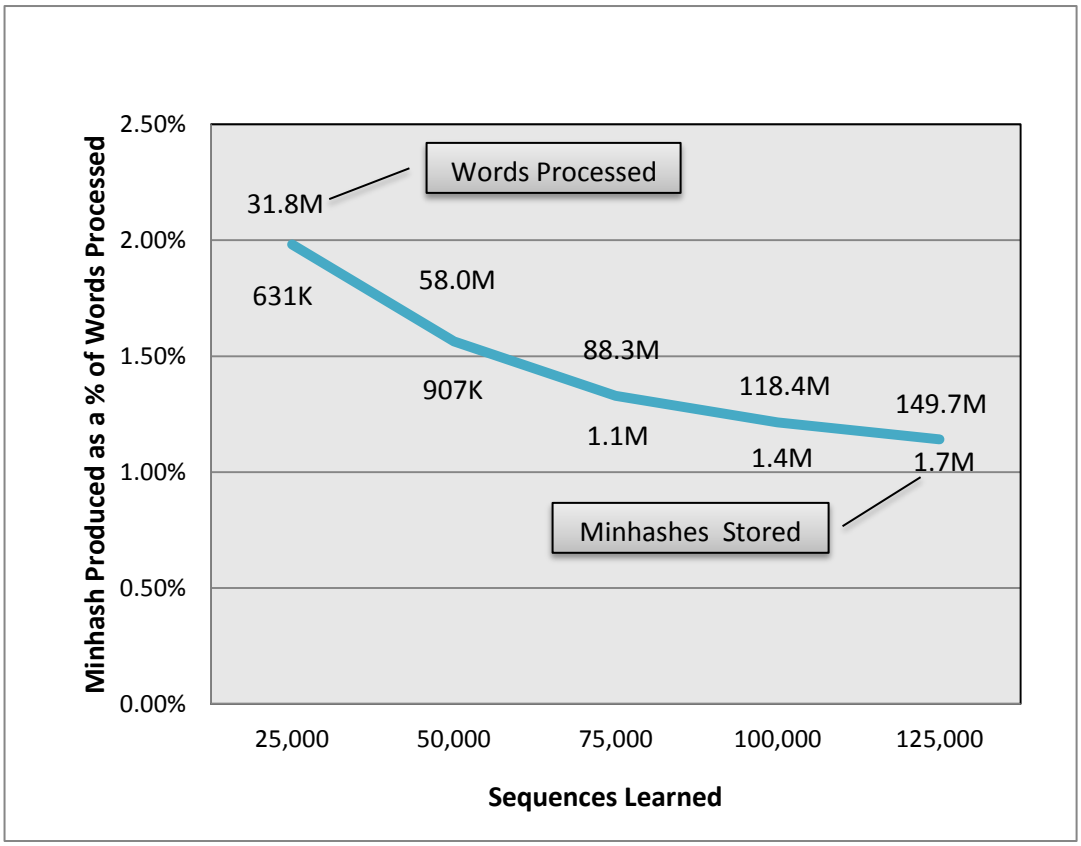


Figure 4.6: Percentage of retained entries in the Strand data structure.

sequences, Strand completes training in 5:39 (mm:ss) while RDP takes 16:50 (mm:ss). The classification time does not vary much with the training set size. Strand's average classification time for the 25,000 sequences is 1:41 (mm:ss) while the average time for RDP is 20:15 (mm:ss).

Finally, we look at memory requirements for Strand. Since we use a word size of 60 bases, there exist $4^{60} \approx 10^{36}$ unique words. For RDP's word size of 8 there are only $4^8 = 65536$ unique words. Strand deals with the huge amount of possible words using minhashing and adding only a small signature for each sequence to the class signature. This means that the Strand data structure will continue to grow as the

volume of training data increases.

The overall space consumption characteristics of Strand are directly related to the selected word size, the minhash signature length, and the amount of sequences learned. Figure 4.6 shows the percentage of unique signature entries (minhash values) stored relative to the number of words processed. With increasing training set size the fraction of retained entries falls from 2% at 25,000 sequences to just above 1% at 125,000 sequences. This characteristic is attributed to the fact that many sequences share words. In total, Strand stores for 125,000 sequences 1.7 million entries which is more than the 65,536 entries stored by RDP, but easily fits in less than 1 GB of main memory which is typically in most modern smart phones.

4.3.4. Conclusion

In this research we have introduced a novel word-based sequence classification scheme that utilizes large word sizes. A highly parallel MapReduce style pipeline is used to simultaneously map gene sequence input data into words, map words into word hashes, reduce all word hashes within a single sequence into a minimum hash signature, and then populates a data structure with category minhash signatures which can be used for rapid classification. Experiments using RDP and Greengenes data show that for 16S rRNA a word size of 60 bases and a sequence minhash signature length of 300 produce the best classification accuracy. Compared to RDP, Strand provides comparable accuracy while performing classification 20 times as fast.

4.4. Using Strand for Abundance Estimation

This work extends our prior presentation of Strand [52], demonstrating its usefulness for performing abundance estimation. We also show how the application achieves comparable accuracy to other abundance estimation programs [121, 157] demonstrat-

ing a more scalable, multi-threaded implementation. This is accomplished using a form of Locality Sensitive Hashing called minhashing. We show the Strand application's ability to scale by replicating classification and training worker processes any number of times across commodity hardware machines. Finally, Strand is benchmarked against the Kraken [157] and CLARK [121] classifiers using the HiSeq and MiSeq training files [157].

4.4.1. Map Reduction Aggregation

For abundance estimation, Strand uses the map reduction aggregation process shown in Figure 4.7 to rapidly prepare and process input data in parallel during training or classification. Map reduction aggregation executes using shared memory during all stages within each Strand worker process. When multiple worker processes are used in a cluster, a single master process combines the outputs from each of the self-contained workers as they complete.

During stage 1 of map reduction aggregation, multiple threads extract words and associated classes from the gene sequence data in parallel. Simultaneously, a stage 2 combiner process minhashes each extracted word eventually creating a minhash signature for each input sequence provided. Finally, the unique minash keys within each minhash signature are summarized by class during the reduce stage. During training, the reduce step adds minhash values into the training data structure, and during classification, minhash values are looked up within the training data structure and minhash intersections for each class are tabulated to determine one or more class similarity estimates.

Strand uses the map reduction aggregation process shown in Figure 4.7 to rapidly prepare and process input data in parallel during training or classification. Map reduction aggregation executes using shared memory during all stages within each Strand worker process. When multiple worker processes are used in a cluster, a single

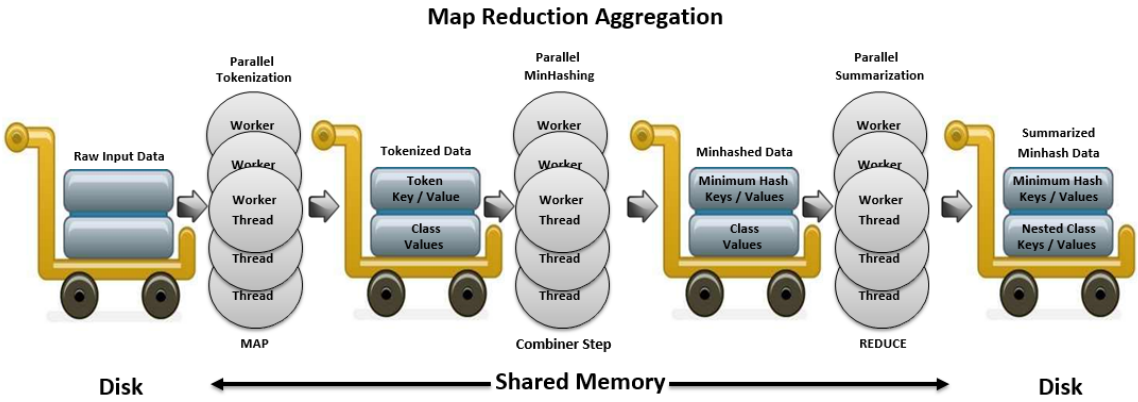


Figure 4.7: Strand Map Reduction Aggregation Processing for a Single Training or Classification Worker Process.

master process combines the outputs from each of the self-contained workers as they complete.

4.4.1.1. Minhashing during Map Reduction Aggregation

Minhashing is utilized within Strand to drastically reduce the amount of storage required for high capacity map reduction aggregation and classification function operations. Map reduction aggregation requires multiple pipeline stages when lossy compression via minhashing is deployed.

In Figure 4.7, Strand uses a map reduction aggregation pipeline including an additional combiner step to facilitate minhashing. This process also represents a more accurate method for Jaccard approximation than mere random selection of words. Minhashing is a form of lossy data compression used to remove a majority of the gene sequence words produced during stage one mapping by compressing all words into a much smaller minhash signature.

During stage one of the map reduction aggregation method shown in Figure 4.7, transitional sequence word outputs are placed into centralized, thread-safe storage

areas accessible to minhash operation workers. In stage 2, a pre-determined number of distinct hashing functions are then used to hash each unique key produced during the stage one map operation one time each. As the transitional keys are repeatedly hashed, only one minimum hash value for each of the distinct hash functions are retained across all keys. When the process is completed only one minimum hash value for each of the distinct hash functions remains in a collection of minhash values which represent the unique characteristics of the learning or classification input data within a minhash signature.

To further enhance minhashing performance, only a single hash function can be used to generate a minhash signature. In this scenario, all words are hashed by a single hashing function and n minimum hash values are selected to make up the minhash signature. This eliminates the overhead of hashing words multiple times to support the family of multiple hashing functions traditionally used to create a minhash signature. Reducing the number of hashing functions however, can come at the expense of additional minhash value collisions and reduced accuracy in certain instances. As discussed in Section 3.4.2 and illustrated in Figure 3.2, this depends on both the number of unique values supported by the hashing function used and the total number of unique gene sequence words contained within the training data corpus and eventually the Strand training data structure. Collisions and reduced accuracy occur when the unique values supported by the hashing function used are not large enough to represent the number of unique words hashed and stored within the training data structure.

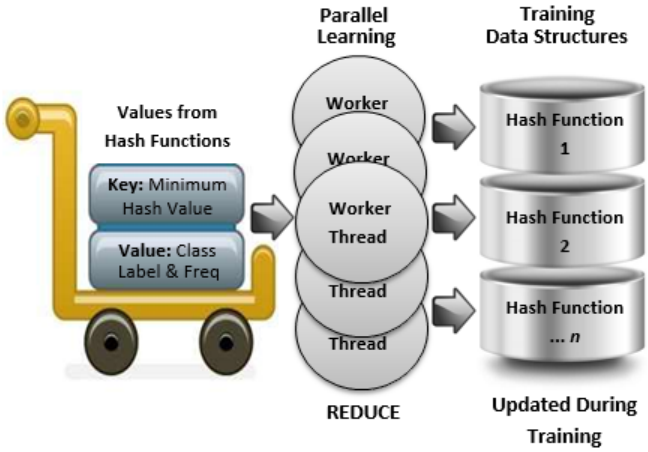


Figure 4.8: The Strand Partitioned Training Data Structure.

The minhash signature is further reduced by storing each minhash value in a partitioned collection of nested categorical key-value pairs. The training data structure illustrated in Figure 4.8 is designed in this manner. The training data structure's nested key value pairs are partitioned or sharded by each distinct hash function used. For example, when the minhashing process uses 100 distinct hash functions to create minhash signatures, the training data structure is divided into 100 partitions. All unique minhash keys created by hash function 0 are stored within partition 0 of the training data structure. Likewise, all unique minhash keys created by hash function 99 are stored in partition 99. However, when only a single hash function is used, no partitions are required.

The partitioned training data structure shown in Figure 4.8 includes minimum hash values which act as the *key* in the nested *categorical* key-value pair collection. Each minhash key contains as it value a collection of the classes which are associated with that key in the system. This collection of classes represents the nested categorical key-value pairs collection. Each nested categorical key-value pair contains a known class as its key and an optional frequency, weight, or any other numerical value which represents the importance of the association between a particular class and the

minhash value key. As discussed further in Section 4.4.3, many binary classification strategies do not require frequencies or other numerical values to be associated with each class. In these instances, only class values may be associated with each minimum hash value key.

4.4.2. Training Data Compression

The keys within the Strand training data structure are compressed by splitting the bits within each numeric key in half dividing them into left and right bit components. When using a 64-bit key for example, this compression splits each 64-bit key into its respective left and right 32-bit components. Compression occurs by grouping together all 64-bit keys with matching left 32-bits and only storing the left 32-bits for all keys with matching values one time.

4.4.2.1. Merge Sort Processing and Training Worker Deduplication

Compression of the Strand training data structure begins using a sort processing step at the end of each training process. Merge sort processing also occurs for each training worker's individual output when multiple worker processes are used in a training cluster. Each nested categorical key value pair collection in the training data structure is sorted by the minhash key. The output from this process can remain in memory or be written out to disk. When multiple training workers are deployed, the merge sort process combines all individual training worker outputs, eliminating duplicate keys and concatenating delimited category values within each record of the merged file.

At this point during processing, category values within the final merge sorted output data may contain duplicates. However, when the training data is loaded into memory for classification, all duplicated category values are removed in a single deduplication step. This single step processing saves substantial overhead when compared

to loading multiple duplicate keys and associated categories into a single training data structure in memory where any number of duplicated key's category collections must be repeatedly merged with a pre-existing collection of unique categories to create a new category set. The cost of concatenating the duplicated key's category collections as a string during merge sort processing turns out to be dramatically faster.

4.4.2.2. 64-Bit Minhash Value Compression

When 64-bit minhash keys are used during training, the Strand training data structure is compressed by storing the left and right 32-bits of each 64-bit key in separate locations. This compression strategy uses the left 32-bits of each 64-bit minhash as the key in the Strand training data structure. The remaining right 32-bits are stored within key's value. Each key's value now contains an array of structs which are made up of all the unique right 32-bits and their respective categories which all share the same left 32-bit key. This technique saves 32-bits of in memory storage each time two 64-bit minhash values share the same left 32-bit key.

Since each of the 64-bit keys and associated categories come from a merge sorted input file, we know that each resulting array of structs containing the right 32-bit keys and associated categories are in right 32-bit key sorted order. During classification, 64-bit keys are split into the left and right 32-bit values. Any time the left 32-bit value contains a value array with length greater than 1, a binary search is executed to determine if the remaining right 32-bits can be located within the array.

Empirical test results on the National Center for Biotechnology Information's (NCBI) RefSeq database [128] show that, on average, this compression technique allows the in memory storage of almost twice as many 64-bit minhash values generated from 31 base gene sequence words.

4.4.2.3. Classification Training Database Optimization

It is important to note that while map reduction aggregation processing for both training and classification should be identical, the training data structures required for learning and classification processing can be different. For example, the parallel processing pipeline used during training requires some form of atomic, thread safe operations for adding and updating minhash keys and associated category values within the training database. During classification however, such thread safety adds unnecessary space in memory and extra processing overhead since only read operations are required.

While 64-bit key compression could be used during training, it is typically not since multiple training workers are deployed. In this scenario, key compression is eliminated in favor of decreased processing time. Memory consumption is still easily managed by simply using a specified number of workers which write all unique 64-bit keys and associated categories to disk when processing is complete. Training memory consumption is managed by using a specified number of training worker processes and input file splits. For example, to decrease the memory required for 5 training workers processing 5 input file splits, a Strand user can simply specify to use 5 workers and 10 input file splits. This effectively cuts the maximum memory required for training in half. The 5 training workers would begin processing input file splits 1-5 having only half of the required input data in memory at any given time. As each worker completes processing, a new training worker is spawned and begins processing the next input file split. Using this configuration, any amount of sequence data can be processed using one or more worker processes on one or more computing devices.

The merge sort process combines each of the intermediary training worker output files into a single file with unique 64-bit keys and associated categories. Loading this entire file into memory is critical for optimal classification processing. The optimized

classification training data structure splits each 64-bit key into left and right 32-bit components compressing all 64-bit keys which share matching left 32-bit components as previously described in Section 4.4.2.2. In instances where all of the classification training data will not fit into memory on a single machine, training data can be partitioned by class or key ranges making a single classification for each partition and then combining results for the final classification.

4.4.3. Classification Function Processing

Using a single training data structure, multiple classification scores are supported. Jaccard Similarity is calculated using the intersection divided by the union between the two sets. No frequency values are required for this similarity measure. For example, the Jaccard similarity between two sets $\mathcal{S}_1$ and $\mathcal{S}_2$ is defined as $S_J(\mathcal{S}_1, \mathcal{S}_2)$, where:

$$S_J(\mathcal{S}_1, \mathcal{S}_2) = \frac{|\mathcal{S}_1 \cap \mathcal{S}_2|}{|\mathcal{S}_1 \cup \mathcal{S}_2|}$$

Weighted Jaccard Similarity is also supported when the class frequency for unique minhash values are retained in the nested categorical key-value pair collection and taken into consideration [76]. The Weighted Jaccard similarity between two sets $\mathcal{S}_1$ and $\mathcal{S}_2$ is defined as $S_{WJ}(\mathcal{S}_1, \mathcal{S}_2)$, where:

$$S_{WJ}(\mathcal{S}_1, \mathcal{S}_2) = \frac{\sum_i min(\mathcal{S}_{1_i}, \mathcal{S}_{2_i})}{\sum_i max(\mathcal{S}_{1_i}, \mathcal{S}_{2_i})}$$

In Strand, Jaccard similarity is approximated by intersecting two sets of minhash signatures where longer signatures provide more accurate Jaccard similarity or distance approximations [129]. Class frequencies may be used to produce other Jaccard Index variations such as the Weighted Jaccard Similarity [76] shown above. However, large performance gains are achieved in Strand by using binary classification

techniques where no nested categorical frequency values or log based calculations are required during classification function operations. In the binary minhash classification approach, minhash signature keys are simply intersected with the minhash keys of known classes to calculate the similarity between a query sequence and a known class.

After minhashing gene sequence words, we only have the sequence minhash signatures $\mathcal{M}_1 = \text{minhash}(\mathcal{S}_1)$ and $\mathcal{M}_2 = \text{minhash}(\mathcal{S}_2)$ representing the two sequences. Fortunately, minhashing [129] allows us to efficiently estimate the Jaccard index using only the minhash signatures:

$$S_J(\mathcal{S}_1, \mathcal{S}_2) \approx \frac{|minhash(\mathcal{S}_1) \cap minhash(\mathcal{S}_2)|}{k},$$

where the intersection is taken hash-wise, i.e., how many minhash values agree between the two signatures.

Next, we discuss scoring the similarity between a sequence minhash signature and the category minhash signatures used for classification. Category signatures are not restricted to k values since they are created using the unique minhash values of all sequence minhash signatures belonging to the category. This is why we do not directly estimate the Jaccard index, but define a similarity measure based on the number of collisions between the minhash values in the sequence signature and the category signature.

Definition 1 (Minhash Category Collision) *We define the Minhash Category Collision between a sequence S represented by the minhash signature $\mathcal{M}$ and a category signature $\mathcal{C}$ as:*

$$\text{MCC}(\mathcal{M}, \mathcal{C}) = |\mathcal{M} \cap \mathcal{C}|,$$

where the intersection is calculated for each minhash hashing function separately.

We calculate MCC for each category and classify the sequence to the category resulting in the largest category collision count.

Many other more sophisticated approaches to score sequences are possible. These are left for future research.

4.4.4. Applying Strand to Machine Learning Tasks

Figure 4.9 shows the self-contained training and classification worker processes in combination with a single training data structure. During training, map reduction aggregation output of known classes are consolidated or further reduced by storing all outputs using the same map reduction aggregation method in a single training data structure. The nested collection of categorical key-value pairs for each key in the training data structure provides a numerical description of that key across any number of categorical keys (i.e. known classes). The training data structure may also be partitioned to support more complex map reduction aggregation outputs or to enhance parallelism during machine learning. A partitioned training data structure was also presented in Figure 4.8 for the purpose of supporting lossy compression using minhashing. The training data structure is updated during training worker processing and read during classification worker processing to apply the specified classification function operations and calculate similarity between a query sequence and one or more known classes within the system.

Unique sequence words are summarized during map reduction aggregation and may optionally calculate the frequency of each unique sequence word identified. The taxonomies associated with each word are made available to the reduce step and act as the associated classes during training. In the naive approach, the training data

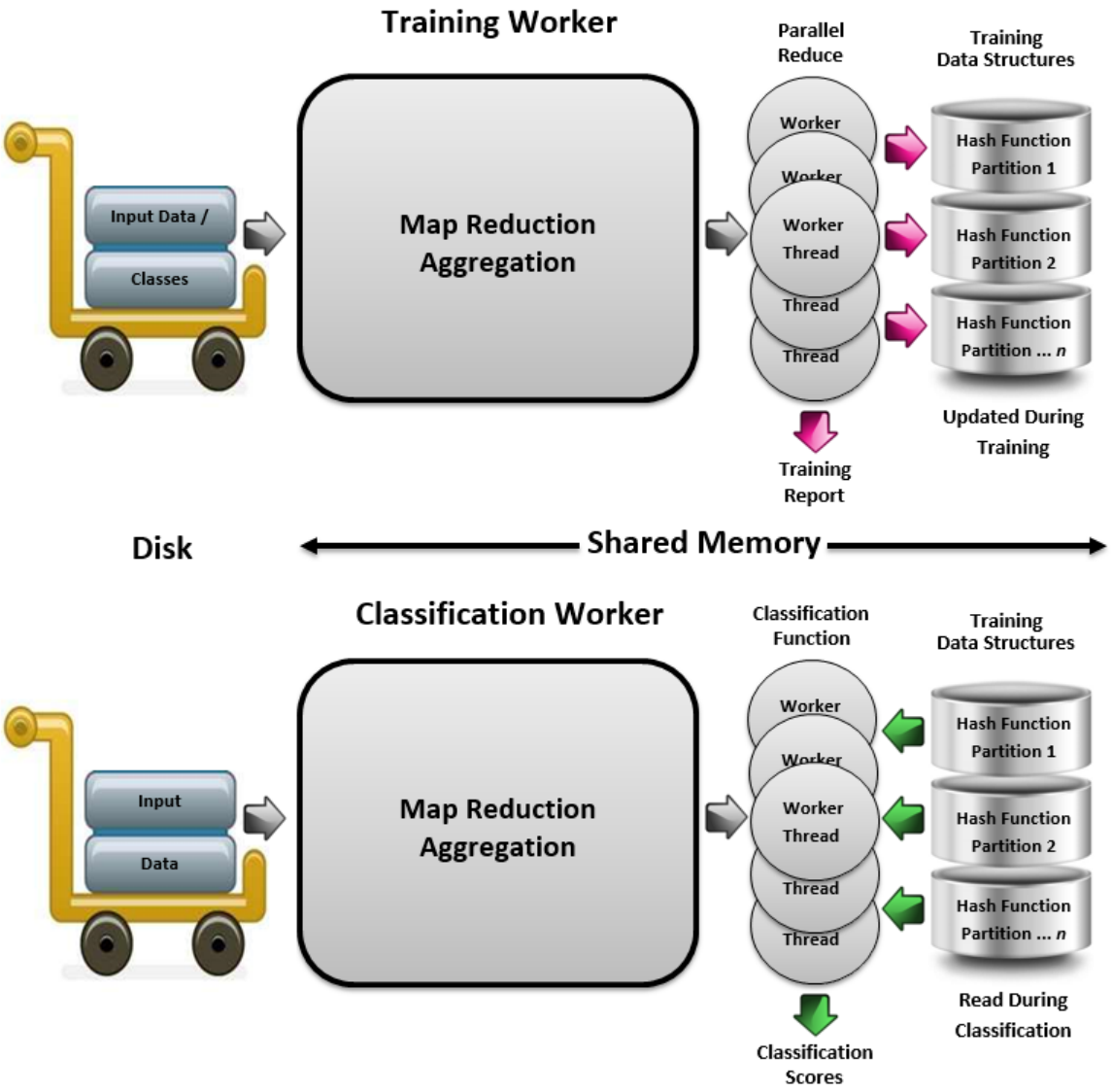


Figure 4.9: Strand Training and Classification Workers using the Collaborative Analytics Framework. This is a copy of Figure 2.13 for reading convenience.

structure contains a single partition where each unique sequence word contains a nested collection of taxonomic classes and the associated frequency for a particular word within each taxonomic class as the nested categorical key-value pair collection. However, since minhashing is used, an additional combiner step is added to hash each word one time retaining the minimum hash value or multiple minimum hash values for each unique hashing function used. These values make up the minhash signature and are stored in a partitioned training data structure with one partition for each unique hashing function or no partitions when a single hashing function is used.

During training, each sequence word extracted is looked up within the training data structure. New training data structure entries are created for any new words identified while the optional frequency value is incremented when a word key already exists. Multiple worker threads are operating at each stage of the map reduction aggregation pipeline in parallel. The map, combiner, and reduce stages within map reduction aggregation operate simultaneously exchanging intermediate data using shared memory within each self-contained training node. This is highly advantageous since all disk I/O is eliminated between each of the map, combiner, and reduce stages. When processing very large volumes of input data, multiple training or classification nodes are deployed to reduce processing times and add additional processing hardware within a Strand computing cluster.

During classification, the same map reduction aggregation method used for training also processes the classification input data. However, a classification function process also calculates a similarity score between the map reduction aggregation outputs and one or more classes stored within the training data structure. When making a multi-class prediction, the class with the highest similarity or the lowest distance is selected. However, the framework is also capable of providing the similarity or distance scores calculated for each of the individual classes. This is useful when input

data may align with multiple classes. For example, a lengthy gene sequence may contain multiple known mutation classes which are highly similar.

Figure 4.9 also shows a Strand classification worker. All stages of map reduction aggregation for the classification and training workers are the same. This is critical since accurate classifications are only made when the summarized data produced by map reduction aggregation for training and classification are created in an identical manner. Outputs produced from the final reduce step are looked up within the training data structure locating matching keys and associated class frequencies as needed. The map reduction aggregation output is processed according to the classification function operations in order to determine a classification score for one or more classes contained within the training data structure.

4.4.5. Strand Computing Clusters

While traditional MapReduce is commonly used for multicore machine learning tasks [30, 61, 62], researchers [60, 101] now recognize the need for parallel machine learning frameworks which strike a balance between the high-level parallel abstractions like MapReduce and lower level multicore development tools.

Gillick et. al. states that an ideal MapReduce style parallel machine learning implementation should provide shared memory to all map tasks on a compute node [62]. Strand takes this idea one step further by exposing shared memory for all processing stages required to create a consistently applied data preparation and aggregation method for the purposes of training or classification during machine learning. In addition, Strand's self-contained training and classification worker processes are easily replicated to scale a machine learning process on any size computer or on any number of commodity hardware machines operating within a cluster.

Both training and classification nodes may be replicated any number of times on either a single server or multiple commodity hardware servers to increase machine

learning throughput. A unique benefit of a Strand machine learning cluster is that classification and training worker processes require no inter-process communication between workers dedicated to the same machine learning task. Once input data is appropriately partitioned each classification and training worker process acts entirely independent of other worker processes within the same framework.

In order to avoid inter-process communication between worker processes, the master process equally divides larger input files into partitions prior to creating additional workers. For each input data partition created, one or more Strand classification or training workers are spawned in separate processes. When the "master" spawns each new process, it specifies command line arguments including the location of the input data and the operation to be performed (i.e. training or classification). The "master" also specifies the output dataset location for each spawned worker and monitors processing until all work is completed.

Once all worker processing is completed, the "master" performs the additional step of combining the outputs of each worker. For example, the "master" combines all training data structures created by one or more training workers into a single consolidated training data structure. In some situations, a RAMDisk partition [19] may be used to avoid loading and unloading training data structure partitions to and from disk. In this scenario, the input data is read once from disk and training data structure outputs are written to the RAMDisk to avoid multiple disk I/O operations and drastically decrease processing times.

Since each training or classification worker is self contained and maintains its own dedicated stack and processing threads, this greatly increases the overall parallelism of the entire application. All MapReduce style processing pipeline data structures experience reduced thread contention for access to intermediate pipeline data structures. In addition, multiple processing nodes can fully utilize processing resources

on a machine which may not be accessible when using only a single process due to process level thread or stack size limitations.

The training data structure includes the consolidation of all processing outputs using similar training or classification methods. Training data structures could also be sharded or partitioned across any number of devices accessible to training or classification workers connected to a distributed network such as a Network Attached Storage (NAS) device, a Hadoop file system, or Spark's Resilient Distributed Datasets.

Within Strand, all master or slave workers run as independent processes. Losing the master does not impact any workers and vice versa. Therefore, when a worker terminates for any reason, a missing output file indicates that an error has occurred. Likewise, when a master terminates, it could simply be restarted. The master can identify any missing outputs and rerun worker processes or combine transitional worker outputs as needed to complete its task.

4.4.6. Results

In this section we first report on a set of experiments demonstrating the optimal performance gained by using a Strand computing cluster even on a single machine. Next, we challenge Strand by training on all known bacterial genomes in the National Center for Biotechnology Information's (NCBI) RefSeq database [128]. This training dataset includes completed microbial genomes from the Bacteria and Archaea kingdoms and is almost 9.5GB in size. Finally, we compare Strand to the Kraken [157] and CLARK [121] abundance estimation classifiers using the microbial metagenomics dataset called "HiSeq" which was presented in [157]. All benchmarks were performed using the Windows 8.1 pro operating system with dual Intel Xeon 2.6Ghz hexacore processors, 24 hyper-threads, and 128GB of RAM.

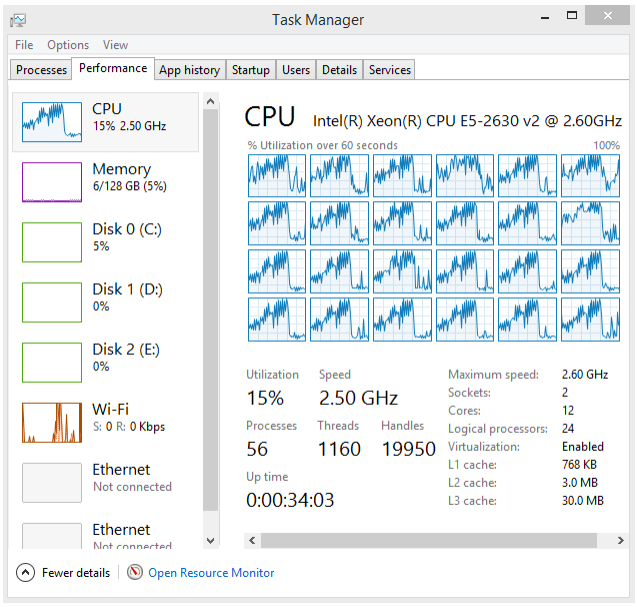


Figure 4.10: CPU utilization for Strand using one Training Worker on 24 Virtual Cores.

4.4.7. Strand Cluster Computing Benefits

One embodiment of Strand is shown in Figure 4.10 executing a single training worker process. Figure 4.10 shows the framework executing 1160 threads with an average CPU utilization of 15%. While the training worker process has ample input data to process, it is limited by the parent process and unable to utilize much of the resources available on the underlying hardware. In this case, processing a 396MB training file takes approximately 7 hours and 35 minutes.

Figure 4.11 illustrates Strand training on the same hardware using ten training workers. The server can clearly be seen operating at 100% CPU utilization while using the same exact input data. In addition, using ten training workers with separate processes effectively doubles the number of executing threads which are increased to 2311 active threads in Figure 4.11. Furthermore, memory utilization is pushed from 6GB to 27GB demonstrating the massive increase in processing throughput. This is most noticeable in the training execution time which decreases to 1 hour and 19

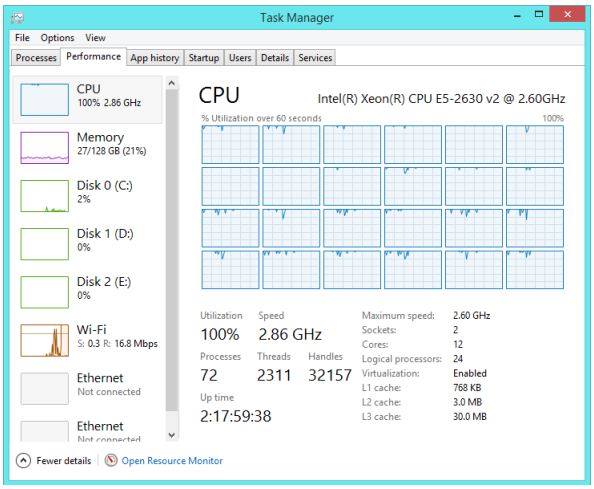


Figure 4.11: CPU utilization for for Strand using Ten Training Workers on 24 Virtual Cores.

minutes. This includes the time it takes the master process to combine all ten of the training datasets produced by each of the ten training workers.

4.4.8. Strand vs. CLARK HiSeq Performance

The Strand application is designed for the Windows operating system. Other operating systems are not supported at this time. While test versions of the application have been ported to Linux using the C# compiler Mono [72], the compiler's support for highly parallel applications is limited and does not perform well on Linux as of Mono version 3.5. Furthermore, the CLARK [121] abundance estimation classifier is written in C++ and not supported on the Windows operating system. CLARK executes using a single process which is highly tuned for optimal performance using the the National Center for Biotechnology Information's (NCBI) RefSeq database [128] as input for training data.

4.4.9. Strand Training on the NCBI Complete RefSeq Database

Performance results for both the Kraken and CLARK Classifiers using the NCBI

RefSeq database [128] have been previously presented [121, 157]. In addition, the CLARK classifier claims to be faster than Kraken [121] during classification and training with similar levels of both precision and sensitivity. For reference, we have included these benchmarks as well in Table 4.3. We also installed CLARK in default mode on a Linux server with 24, Intel(R) Xeon(R) CPU X5690 @ 3.47GHz and 198GB of RAM. Since CLARK produces a list of all .fna files ("targets.txt") used during training, installing CLARK on our own linux server enabled training Strand using exactly the same RefSeq input files CLARK used in default mode.

We now compare Strand to the CLARK classifier when using the National Center for Biotechnology Information's (NCBI) RefSeq database [128]. The results for Strand presented below, used sequences and assigned taxonomy id's from each of the files contained within the CLARK "targets.txt" file. The benchmarks for CLARK using these files are listed in Table 4.3 as "CLARK Default Mode". Other results for both CLARK and Kraken were obtained from Table S1 in [122]. While this is certainly not an ideal comparison, these are reasonable results to review considering that the three classifiers cannot execute on the same operating system.

RefSeq Training Benchmarks			
Classifier Version	Train Time (hh:mm)	Disk Size GB	Peak RAM
Strand 10% Sample	01:37:46	13.3	119
Strand 20% Sample	02:24:17	26.7	121
Strand 30% Sample	03:13:52	40.1	127
CLARK Default Mode	01:46:00	30	31.56
CLARK	02:45:00	42.4	167.9
Kraken	06:07:00	141.0	164.1

Table 4.3: Comparison of Strand, CLARK, and Kraken when training on the Refseq Database.

Table 4.3 shows training times for Strand, CLARK default mode, CLARK, and Kraken when using the NCBI RefSeq training data. Strand demonstrates reduced

training times for minhash value sample sizes up to 30% and also uses less space on disk to save the training models created. During training, Strand has a substantially reduced memory footprint. As previously mentioned, the memory footprint is also entirely parameter configurable by specifying the number of worker processes and input file splits used on a given machine. While Strand training times do increase as input file splits increase, it is important to note that all of the training times reflected in Table 4.3 were produced using dual Intel Xeon 2.6Ghz hexacore processors, 12 cores, 24 hyper-threads, and 128GB of RAM. With more memory available and less the training input file splits, these training times could be substantially reduced. The benchmarks taken for both CLARK and Kraken were executed on a Dell PowerEdge T710 server using dual Intel Xeon X5660 2.8 Ghz processor chips, 12 cores, and 192 GB of RAM [121].

One key advantage Strand has over other classifiers is the ability to simply alter the minhash sample size for any given training or classification execution. Both Kraken and CLARK require multiple versions of their software which are tailored for various features such as faster processing or a smaller training database footprint. Changing the configurable worker processes, input file splits, and minhash sample sizes, gives Strand users ultimate flexibility in managing the training database in memory footprint, execution times, precision, and sensitivity levels. All strand training results shown in Table 4.3 were produced using 15 Strand training workers and 50 training input file splits of the NCBI RefSeq database. The training input file splits are produced by providing all of the 4,345 NCBI RefSeq .fna files listed within the CLARK "targets.txt" file as input to Strand. Strand then divides the file list into 50 equal parts for training worker processing. Strand can also be simply pointed at a directory of .fna files for training. Each NCBI GI number is resolved from within each .fna file and taxonomy id's are assigned to the training data at any taxonomy

level. The input data file paths are then divided into equal input file splits.

subsectionStrand Classifying the"HiSeq" Dataset

The Strand and CLARK classifiers were used to classify the microbial metagenomics dataset called HiSeq [157] after training on the NCBI RefSeq database. CLARK was installed in default mode on a Linux server with 24, Intel(R) Xeon(R) CPU X5690 @ 3.47GHz and 198GB of RAM. The classification times listed in Table 4.4 represent lowest time achieved out of three runs. Initial results show both Stand and CLARK performing with comparable precision at 98.3% and 98.7% respectively. Both classifiers performed the 10,000 HiSeq classifications in under 2 seconds for all configurations shown in Table 4.4.

HiSeq Classification Benchmarks					
Classifier	Load Time	Classify Time	Precision	Sensitivity	RAM GB
Strand 10%	00:13:58	1 sec	98.5%	66.8%	43
Strand 20%	00:33:26	1 sec	98.3%	69.3%	82
Strand 30%	01:19:18	00:01:36	98.1%	69.9%	127
CLARK Default	00:01:42	$<$ 1 sec	98.7%	70.5%	70.1

Table 4.4: Comparison of Strand, CLARK, and Kraken when classifying the HiSeq microbial metagenomics dataset. NOTE: The Strand 30% Sample left $<$ 1GB of memory for classification and likely caused disk thrashing witch degraded processing time.

4.4.10. Conclusion

We have presented Strand (The Super Threaded Reference-Free Alignment-Free Nsequence Decoder) which is a novel system and method for the shotgun classication of gene sequence data into any number of associated categories or gene sequence taxonomies using highly scalable, multicore training and classification workers to exploit shared memory, parallel processing pipelines and efficiently train and classify input data. We demonstrate Strand's effective use in high performance computing clusters

to process massive volumes of input data while still maintaining a very small training database footprint and making classification predictions with very high sensitivity and precision. Experimental results show that Strand is able to scale well using large amounts of computing resources and perform classification on smaller laptop devices using its full training database.

Chapter 5

DEVELOPING FEATURES FOR CYBERCRIME

I now provide a detailed discussion on developing features for cybercrime. This chapter first explores features to identify replicated criminal websites which are loose copies of one another that have been duplicated by criminals involved with both Ponzi Schemes and Escrow Fraud. These features are used for an unsupervised machine learning technique called Optimized Combined Clustering which is presented in Chapter 6. Next, a feature extraction framework for the identification of websites selling counterfeit goods is discussed. In Chapter 7, websites selling counterfeit goods are analyzed and classified using supervised machine learning techniques and the features presented in Section 5.2.

5.1. Developing Features to Identify Replicated Criminal Websites

To be successful, cybercriminals must figure out how to scale their scams. They duplicate content on new websites, often staying one step ahead of defenders that shut down past schemes. For some scams, such as phishing and counterfeit-goods shops, the duplicated content remains nearly identical. In others, such as advanced-fee fraud and online Ponzi schemes, the criminal must alter content so that it appears different in order to evade detection by victims and law enforcement. Nevertheless, similarities often remain, in terms of the website structure or content, since making truly unique copies does not scale well.

In this section, we present automated methods to extract key website features, including rendered text, HTML structure, file structure and screenshots. We describe a

process to automatically identify the best combination of such attributes to most accurately cluster similar websites together in Chapter 6. To demonstrate the method's applicability to cybercrime, we evaluate its performance against two collected datasets of scam websites: fake-escrow services and high-yield investment programs (HYIPs). We show that our method more accurately groups similar websites together than does existing general-purpose consensus clustering methods using the features we describe in this chapter.

5.1.1. Process for Identifying Replicated Criminal Website Features

We now describe a general-purpose method for identifying the best features to cluster together replicated criminal websites. We provide a high-level overview, which is now briefly presented before each step is discussed in greater detail in the following sections. The implementation of optimized combined clustering for these websites is also discussed in detail within Chapter 6.

Steps for Developing Features to Identify Replicated Criminal Websites:

1. **URL Crawler:** Raw information on websites is gathered.
2. **URL Feature Extraction:** Complementary attributes such as website text and HTML tags are extracted from the raw data for each URL provided.
3. **Input attribute feature files:** Extracted features for each website are saved into individual features files for efficient pairwise distance calculation.
4. **Distance matrices:** Pairwise distances between websites for each attribute are computed using Jaccard distance metrics.

Step 1 is described in Section 5.1.2. Steps 2 and 3 are described in Section 5.1.7, while step 4 is described in Section 5.1.8.

5.1.2. Data Collection Methodology

In order to demonstrate the generality of our clustering approach discussed in Chapter 6, we collect datasets on two very different forms of cybercrime: online Ponzi schemes known as High-Yield Investment Programs (HYIPs) and fake-escrow websites. In both cases, we fetch the HTML using `wget`. We followed links to a depth of 1, while duplicating the website's directory structure. All communications were run through the anonymizing service Tor [44].

Data Source 1: Online Ponzi schemes We use the HYIP websites identified by Moore et al. in [114]. HYIPs peddle dubious financial products that promise unrealistically high returns on customer deposits in the range of 1–2% interest, compounded *daily*. HYIPs can afford to pay such generous returns by paying out existing depositors with funds obtained from new customers. Thus, they meet the classic definition of a Ponzi scheme. Because HYIPs routinely fail, a number of ethically questionable entrepreneurs have spotted an opportunity to track HYIPs and alert investors to when they should withdraw money from schemes prior to collapse. Moore et al. repeatedly crawled the websites listed by these HYIP aggregators, such as `hyip.com`, who monitor for new HYIP websites as well as track those that have failed. In all, we have identified 4 191 HYIP websites operational between November 7, 2010 and September 27, 2012.

Data Source 2: Fake-escrow websites A long-running form of advanced-fee fraud is for criminals to set up fraudulent escrow services [113] and dupe consumers with attractively priced high-value items such as cars and boats that cannot be paid for using credit cards. After the sale the fraudster directs the buyer to use an escrow service chosen by the criminal, which is in fact a sham website. A number of volunteer groups track these websites and attempt to shut the websites down by notifying hosting providers and domain name registrars. We identified reports from two lead-

ing sources of fake-escrow websites, `aa419.org` and `escrow-fraud.com`. We used automated scripts to check for new reports daily. When new websites are reported, we collect the relevant HTML. In all, we have identified 1 216 fake-escrow websites reported between January 07, 2013 and June 06, 2013.

For both data sources, we expect that the criminals behind the schemes are frequently repeat offenders. As earlier schemes collapse or are shut down, new websites emerge. However, while there is usually an attempt to hide evidence of any link between the scam websites, it may be possible to identify hidden similarities by inspecting the structure of the HTML code and website content. We next describe a process for identifying such similarities.

5.1.3. Feature Extraction Processing

Using a highly parallel producer-consumer style pipeline program called "the creeper"; each targeted website's structure was interrogated extracting HTML, displayed text, file names, and screen shots. The extracted data was then stored in individual directories and files at both the webpage and entire website level (when applicable). For example, website level HTML files would contain all HTML for all webpages in an entire website within one text file named www.websitename.com.txt.

Further processing was then performed by "the creeper" to extract the more targeted features from this high-level stage 1 input data. Performing the stage 1 high-level data organization allowed us to test the accuracy of several different feature data extraction methods prior to selecting three targeted features for clustering which were the optimal predictors of our ground truth data. Stage 1 high-level data extractions were first performed using a custom "headless" browser adapted from the Watin package for C#[1]. However, when screen shots were collected, eight Selenium Firefox

[1] http://www.watin.org

browsers were executed in parallel for this task [2].

5.1.4. Selecting an Appropriate Distance Metric

Candidate features were written to files at the website and webpage level comprising separate directories for each candidate feature. All directories, files, and file naming conventions are arranged in a manner conducive to performing set based distance metric calculations between all files within a particular directory.

Given this arrangement an infinite number of distance metric calculations can be performed by considering each file within the directory as a set and each line within the file as an item within the set. Set intersections and unions can easily be performed by loading file contents into memory for set comparisons. Furthermore, dot products and magnitudes can be efficiently calculated making the calculation of both Jaccard and Cosine distances between files performance optimal.

The proposed arrangement also facilitates processing all files and performing such calculations in parallel using the map reduce style framework previously described in Chapter 2. The high-level data organization also allows for more intensive distance based calculations such as Edit Distance using portions of the displayed text, HTML, or File names.

Jaccard, Cosine and Edit Distance metrics were deployed by the creeper program to create input attribute distance matrices using each input attribute directory and calculating the distances between all files included within the targeted directory.

5.1.5. Map Reducing Distance Matrices

The creeper program includes map reduce style processing pipelines which are provided for the rapid parallel processing and classification of extremely large volumes of targeted data which allows programmers to create and deploy application

[2] http://www.seleniumhq.org/

specific map reduction aggregation methods and classification metric functions utilizing various distance metrics. The map reduction aggregation methods specify how targeted input data will be aggregated within the current system.

Targeted input data is consistently dissected by the mapping processes into individual, independent units of intermediate work typically comprising consistently mapped data keys and values that are conducive to simultaneous parallel reduction processing. The reduction methods continually and simultaneously aggregate or reduce the mapped data keys and values by eliminating the matching keys and aggregating the corresponding values consistent with the reduction operations for all matching keys which are encountered during map reduction processing.

In the case of map reducing website sentences, sentences extracted from each website's displayed text are accessed and read into memory from each file within the displayed website text sentences directory. A separate background thread is then used to process each sentence set in parallel creating matches that require distance calculations between all website sentence files contained within the sentences directory. Each "match" is placed into a thread-safe blocking collection where a secondary background process manages threads which are continually consuming the matches and performing the distance calculations between sentence files using the specified distance metric (in this case Jaccard or Cosine Distance). Finally, matches are organized into a distance matrix and each distance matrix line is written to comma delimited file which can easily be imported into R for clustering processes as described in Chapter 2.

5.1.6. Identifying and Extracting Website Features

We identified four primary features of websites as potential indicators of similarity: displayed text, HTML tags, directory file names, and image screenshots. These are described in Section 5.1.7. In Section 5.1.8 we explain how the features are computed in a pairwise distance matrix.

5.1.7. Website Features

Website Text To identify the text that renders on a given webpage, we used a custom "headless" browser adapted from the Watin package for C#[3]. We extracted text from all pages associated with a given website, then split the text into sentences using the OpenNLP sentence breaker for C#. Additional lower level text features were also extracted such as character n-grams, word n-grams, and individual words for similarity benchmarking. All text features were placed into individual bags by website. Bags for each website were then compared to create pairwise distance matrices for clustering.

HTML Content Given that cybercriminals frequently rely on kits with similar underlying HTML structure [117], it is important to check the underlying HTML files in addition to the rendered text on the page. A number of choices exist, ranging from comparing the document object model (DOM) tree structure to treating tags on a page as a set of values. From experimentation, we found that DOM trees were too specific, so that even slight variations in otherwise similar pages yielded different trees. We also found that sets of tags did not work well, due to the limited variety of unique html tags. We found a middle way by counting how often a tag was observed in the HTML files.

All HTML tags in the website's HTML files were extracted, while noting how many times each tag occurs. We then constructed a compound tag with the tag name and its frequency. For example, if the "
" tag occurs 12 times within the targeted HTML files, the extracted feature value would be "
12".

[3] http://www.watin.org

File Structure We examined the directory structure and file names for each website since these could betray structural similarity, even when the other content has changed. However, some subtleties must be accounted for during the extraction of this attribute. First, the directory structure is incorporated into the filename (e.g., `admin/home.html`). Second, since most websites include a home or main page given the same name, such as `index.htm`, `index.html`, or `Default.aspx`, websites comprised of only one file may in fact be quite different. Consequently, we exclude the common homepage file names from consideration for all websites. Unique file names were placed into bags by website and pairwise distances were calculated between all websites under consideration.

Website Screenshot Images Finally, screenshots were taken for each website using the Selenium automated web browser for C#[4]. Images were resized to 1000 x 1000 pixels. We calculated both vertical and horizontal luminosity histograms for each image. Image luminosity features and similarity measures were determined using the Eye.Open image library for C#[5]. During image feature extraction, the red, green, and blue channels for each image pixel were isolated to estimate relative luminance, and these values were then aggregated by each vertical and horizontal image pixel row to calculate two luminosity histograms for each image.

5.1.8. Constructing Distance Matrices

For each input attribute, excluding images, we calculated both Jaccard and Cosine distances between all pairs of websites creating pairwise distance matrices for each input attribute and distance measure. During evaluation it was determined that Jaccard distance was the most accurate metric for successfully identifying criminal

[4] http://www.seleniumhq.org/
[5] https://similarimagesfinder.codeplex.com/

website replications.

The Jaccard distance between two sets S and T is defined as 1- $J(S,T)$, where:

$$J(S,T) = \frac{|S \cap T|}{|S \cup T|}$$

Consider comparing website similarity by sentences. If website A has 50 sentences in the text of its web pages and website B has 40 sentences, and they have 35 in common, then the Jaccard distance is $1 - J(A,B) = 1 - \frac{35}{65} = 0.46$.

Website screenshot images were compared for both vertical and horizontal similarity using luminosity histograms. The luminosity histograms for each matched image pair were compared for similarity by calculating the weighted mean between both the vertical and horizontal histograms. Next, both the average and maximum similarity values between histograms were empirically evaluated for clustering accuracy. Taking the average similarity score between the vertical and horizontal histograms performed best during our evaluation. Once the average vertical and horizontal similarity score was determined, then the pairwise image distance was calculated as 1 - the pairwise image similarity.

Distance matrices were created in parallel for each input attribute by "mapping" website input attributes into pairwise matches, and then simultaneously "reducing" pairwise matches into distances using the appropriate distance metric. The pairwise distance matrices were chosen as output since they are required input for the hierarchical agglomerative clustering process used during the optimized clustering process presented in Chapter 6.

5.2. Developing Features to Identify Websites Selling Counterfeit Goods

We now present a methodology for collecting data on the prevalence of counterfeit goods in web search. We build an accurate classifier using features automatically

extracted from website content that distinguishes legitimate from fake sellers based upon data returned by search results in Chapter 7. We investigate the practice of websites selling counterfeit goods. We inspect web search results for 225 queries across 25 brands. In Chapter 7, we also devise a binary classifier that predicts whether a given website is selling counterfeits by examining automatically extracted features such as WHOIS information, pricing and website content. We then apply the classifier to results collected between January and August 2014. We find that, overall, 32% of search results point to websites selling fakes.

For 'complicit' search terms, such as "replica rolex", 39% of the search results point to fakes, compared to 20% for 'innocent' terms, such as "hermes buy online". Using a linear regression, we find that brands with a higher street price for fakes have higher incidence of counterfeits in search results, but that brands who take active countermeasures by filing DMCA requests experience lower incidence of counterfeits in search results. Finally, we study how the incidence of counterfeits evolves over time, finding that the fraction of search results pointing to fakes remains remarkably stable.

In the following sections we describe the data collection methodology and features used to classify websites selling counterfeit goods. In Chapter 7, we use these features to identify counterfeit retailers and explore how such features can be used to identify and combat the sale of counterfeit goods.

5.2.1. Data Collection Methodology

Researchers have extensively studied the online sale of unlicensed pharmaceuticals [90, 94, 100, 105], as well as the unauthorized acquisition of digital goods such as music, movies and software [81, 137]. We decided to focus instead on the sale of counterfeit consumer physical goods, due to its prevalence and relative lack of attention from the research community.

5.2.2. Constructing Search Queries

We first had to decide on which brands to focus our investigation. We began by collecting data on five seed brands: Ugg, Coach, Rolex, Hermes, and Oakley. After gathering search results on associated queries, we scraped all product listing pages from 68 stores manually identified as selling counterfeits. We decided to focus on the 25 most observed brands in the inventories of the confirmed counterfeit stores.

We are very interested in measuring the extent to which consumers intending to buy from authorized retailers are instead presented with links to websites selling fakes. To that end, the 25 brands were then paired with search terms of varying levels of "innocence". We selected three search terms for each innocence level: innocent, grey, and complicit. We deem as innocent the keywords "fast delivery," "buy online" and the lack of keyword (meaning the query is only the product name). We deem the keywords "replica," "fake," and "knockoff" as complicit, as any shopper using those terms is clearly seeking out counterfeited goods. Between these extremes lie grey keywords "cheap," "discount," and "sale", since there is ambiguity as to the intention of the shopper. In total, we combine all 25 brands with each of the 9 keywords to yield 225 unique search queries.

5.2.3. Gathering Data on Websites in Search Results

We automatically issued queries to the Google Custom Search API to obtain the top 100 results for each of the 225 terms.[6] We then visited the URLs using an automated browser, storing the HTML and a screenshot.

When the automated browser visited the search results, a script checked for the presence and properties of elements we believe to be indicative of malicious intent.

[6]We note that the results returned by the Custom Search API are not identical to those obtained via Google's website. Because Google's Terms of Service prohibit automated collection of search results from its website, we elected to use the API instead. Given the advent of personalized search, it is impossible to collect the exact search results that will be presented to all users.

The elements observed were IFrames, currencies, and pricing information.

We removed price outliers triggered by parsing errors or ambiguities in brand names (e.g., searches for Coach purses can also return advertisements for buses). We also converted non-US currency to US dollars for consistency.

Pricing data was found using regular expressions and a combination of currency symbols, and price associations were found (for example a product might contain both an "original price" and a "discount price") by locating individual prices and climbing the DOM structure. We also fetched the WHOIS data for each website, extracting the date and country of registration, as well as whether or not a privacy or proxy registration had been used [34].

We note that many shops attempt to hide their true nature to search engines by "cloaking" or by redirecting from a hacked website to a shop selling fakes. We deal with this problem by inspecting the destination website as presented to a browser. As described in the next section, we do build features that indirectly check for this malicious behavior, e.g., by also visiting the top-level page that is unlikely to be cloaked or to redirect elsewhere.

We gathered data in two distinct periods. An initial study carried out in January 2014 identified 21 646 search results and 6 979 distinct websites. To inspect for changes in behavior over time, we reissued the same queries weekly between June and August 2014.

5.2.4. Feature Selection and Extraction

We constructed features after inspecting many websites selling counterfeit goods. We focus on three classes of features: URL-level, page-level, and website-level features.

URL-level features The least resource-intensive approach is to select features based

upon only the URL. This approach has been used to identify phishing websites [21] and malware [102].

1. **Replica in FQDN**

   This Boolean feature identifies when the term "replica" is contained as part of the web page's fully qualified domain name (FQDN). We also considered the term "knockoff", but that was often associated with articles and blogs decrying counterfeits.

2. **Length of FQDN**

   This numeric feature denotes the number of characters present in the URL's fully qualified domain name. Websites selling fakes frequently use subdomains concatenating several words.

Page-level features We also inspected the scraped HTML content to identify additional features indicating that counterfeits may be sold.

1. **Number of Currencies Seen**

   Unlike most big box retailers which offer custom websites for each country serviced, counterfeit goods websites typically offer a single unified site which is designed to service any number of countries. Furthermore, it is uncommon for these sites to implement their own custom payment vehicles. Most of the third party payment solutions deployed offer checkout alternatives which include providing payment in a large variety of currencies.

2. **Large IFrames**

   Unusually positioned and sized IFrames are used to obfuscate malicious scripts and redirections common in criminal websites [27,104]. We define large IFrames as having unusually large height and width in addition to containing a different top level domain which is also not part of the Alexa top 1000 domains [1].

3. **Percent Savings Average**

   This numeric feature indicates the average percentage of savings on a given webpage. This is relevant on online stores whose pricing data was able to be scraped automatically, in the case where two prices were listed in association with at least one item (an "original" price and their price, to demonstrate the savings). The average of the savings percentages on a given page is stored in the hopes of finding counterfeit stores offering ludicrously high savings.

4. **Number of Times Duplicate Price Seen**

   This numeric feature specifies the number of times a duplicate price was seen on a given page. For example, a page with various products listed at prices $40, $45, $40, $50, $40, and $45 has 3 duplicate prices present ($40 is repeated twice and $45 is repeated once). This feature was included to catch lazy counterfeit store owners who copy and paste products, changing titles but not prices.

5. **Page Contains Webmail Address**

   This Boolean feature identifies when a webpage contains an email address which includes the text "@yahoo.com", "@gmail.com", or "@hotmail.com". It would be highly unusual for a legitimate brand reseller to utilize a free webmail account.

6. **Unique Brand Mention Count**

   The unique brand mention count represents the number of unique brand mentions identified within the webpage's HTML content. Counterfeit websites often stuff their pages with multiple brand mentions, in hopes of promoting the website in search results.

7. **Top-Level Page Mentions Brand**

   We also visited the top-level web page to look for the mention of any brand (a string search for any of the brands in the root page's text). It indicates that a website may have been hacked if the top-level page makes no mention of any

brands while the page listed in the search results does. A website hacked to host or redirect to a store is almost certainly selling counterfeits.

8. **Content Consistent with Takedown Page**
   This Boolean feature identifies when a webpage has been taken down and replaced with content from brand-enforcement companies.

Website-level features The final category of features were those describing characteristics of the website itself, as opposed to the displayed content.

1. **Private or China-registered WHOIS**
   While legitimate companies use private and proxy WHOIS registrations [35], it is frequently employed by those conducting dubious operations such as selling counterfeit goods. Furthermore, we observed that many websites selling replicas have operations based in China.

2. **WHOIS Registration Age Under 1 Year**
   This Boolean feature identifies when a webpage is less than one year old. Counterfeit websites often rely on newly registered domains, which replace older ones that have been suspended.

3. **Website In Alexa Top 100K**
   This Boolean feature identifies when a webpage is ranked in the top one hundred thousand websites by Alexa based on the webpage's web traffic. We expect counterfeit pages to draw in less traffic than licensed retailers.

5.3. Conclusion

In the following two chapters, we present research which utilizes the features discussed in this chapter to solve real world problems related to Cybercrime. In Chapter 7, we build supervised classifiers to detect websites selling counterfeit goods

using Logistic Regression, Adaptive Boosting, and Support Vector Machine models. We demonstrate that such features can successfully identify these criminal websites with high levels of accuracy. Chapter 6 applies the features discussed in Section 5.1 to identify repeat offenders creating loose copies of both Ponzi Scheme and Escrow Fraud related websites. Each of these chapters also utilize implementations of the Collaborative Analytics Framework to produce highly efficient feature sets for Cybercrime related data.

Chapter 6

IDENTIFICATION OF PONZI SCHEME AND ESCROW FRAUD WEBSITES

6.1. Introduction and Background

Cybercriminals have adopted two well-known strategies for defrauding consumers online: large-scale and targeted attacks. Many successful scams are designed for massive scale. Phishing scams impersonate banks and online service providers by the thousand, blasting out millions of spam emails to lure a very small fraction of users to fake websites under criminal control [58, 112]. Miscreants peddle counterfeit goods and pharmaceuticals, succeeding despite very low conversion rates [82]. The criminals profit because they can easily replicate content across domains, despite efforts to quickly take down content hosted on compromised websites [112]. Defenders have responded by using machine learning techniques to automatically classify malicious websites [127] and to cluster website copies together [13, 89, 95, 152].

Given the available countermeasures to untargeted large-scale attacks, some cybercriminals have instead focused on creating individualized attacks suited to their target. Such attacks are much more difficult to detect using automated methods, since the criminal typically crafts bespoke communications. One key advantage of such methods for criminals is that they are much harder to detect until after the attack has already succeeded.

Yet these two approaches represent extremes among available strategies to cybercriminals. In fact, many miscreants operate somewhere in between, carefully replicating the logic of scams without completely copying all material from prior iterations of

the attack. For example, criminals engaged in advanced-fee frauds may create bank websites for non-existent banks, complete with online banking services where the victim can log in to inspect their "deposits". When one fake bank is shut down, the criminals create a new one that has been tweaked from the former website. Similarly, criminals establish fake escrow services as part of a larger advanced-fee fraud [113]. On the surface, the escrow websites look different, but they often share similarities in page text or HTML structure. Yet another example is online Ponzi schemes called high-yield investment programs (HYIPs) [114]. The programs offer outlandish interest rates to draw investors, which means they inevitably collapse when new deposits dry up. The perpetrators behind the scenes then create new programs that often share similarities with earlier versions.

The designers of these scams have a strong incentive to keep their new copies distinct from the old ones. Prospective victims may be scared away if they realize that an older version of this website has been reported as fraudulent. Hence, the criminals make a more concerted effort to distinguish their new copies from the old ones.

While in principle the criminals could start over from scratch with each new scam, in practice it is expensive to recreate entirely new content repeatedly. Hence, things that can be changed easily are (e.g., service name, domain name, registration information). Website structure (if coming from a kit) or the text on a page (if the criminal's English or writing composition skills are weak) are more costly to change, so only minor changes are frequently made.

The purpose of this paper is to design, implement, and evaluate a method for clustering these "logical copies" of scam websites. Section 6.2 gives a high-level overview of the combined clustering process. In Section 5.1.2 we describe two sources of data on scam websites used for evaluation: fake-escrow websites and HYIPs. Next, Sec-

tion 5.1.6 details how individual website features such as HTML tags, website text, file structure and image screenshots are extracted to create pairwise distance matrices comparing the similarity between websites. In Section 6.3 we outline two optimized combined clustering methods that takes all website features into consideration in order to link disparate websites together. We describe a novel method of combining distance matrices by selecting the minimum pairwise distance. We then evaluate the method compared to other approaches in the consensus clustering literature and cybercrime literature to demonstrate its improved accuracy in Section 6.4. In Section 6.5 we apply the method to the entire fake-escrow and HYIP datasets and analyze the findings. We review related work in Section 6.6 and conclude in Section 6.7.

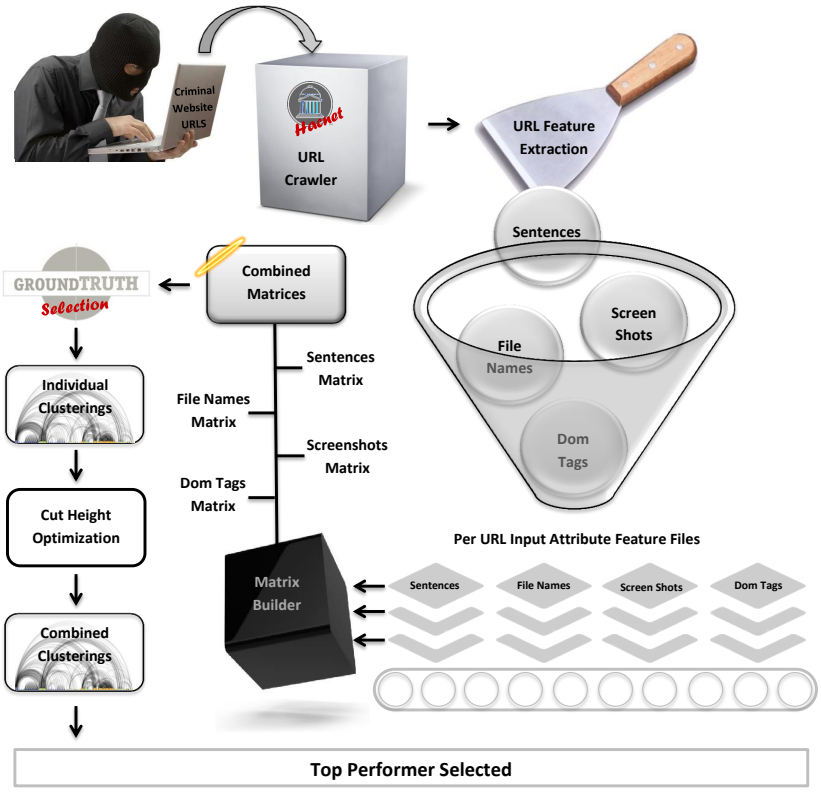


Figure 6.1: High-level diagram explaining how the method works.

6.2. Process for Identifying Replicated Criminal Websites

This paper describes a general-purpose method for identifying replicated websites. Figure 6.1 provides a high-level overview, which is now briefly described before each step is discussed in greater detail in the following sections.

1. **URL Crawler:** Raw information on websites is gathered.
2. **URL Feature Extraction:** Complementary attributes such as website text and HTML tags are extracted from the raw data for each URL provided.
3. **Input attribute feature files:** Extracted features for each website are saved into individual features files for efficient pairwise distance calculation.
4. **Distance matrices:** Pairwise distances between websites for each attribute are computed using Jaccard distance metrics.
5. **Individual Clustering:** Hierarchical, agglomerative clustering methods are calculated using each distance matrix, rendering distinct clusterings for each input attribute.
6. **Combined Matrices:** Combined distance matrices are calculated using various individual distance matrix combinations.
7. **Ground Truth Selection:** Criminal websites are manually divided into replication clusters and used as a source of ground truth.
8. **Cut Height Optimization:** Ground truth clusters are used in combination with the Rand Index to identify the optimal clustering cut height for each input attribute.
9. **Combined Clustering:** Hierarchical, agglomerative clustering methods are calculated using each combined distance matrix to arrive at any number of multi-feature clusterings.

10. **Top Performer Selection:** The Rand Index is calculated for all clusterings against the ground truth to identify the top performing individual feature or combined feature set.

Step 1 is described in Section 5.1.2. Steps 2 and 3 are described in Section 5.1.7, while step 3 is described in Section 5.1.8. Finally, the clustering steps (5–10) are described in Section 6.3.

6.3. Optimized Combined Clustering Process

Once we have individual distance matrices for each input attribute as described in the previous section, the next step is to build the clusters. We first describe two approaches for automatically selecting cut-heights for agglomerative clustering: *dynamic cut height*, which is unsupervised, and *optimized cut height*, which is supervised. Next we compute individual clusterings based on each input attribute. Finally, we construct combined distance matrices for combinations of input attributes and cluster based on the combined matrices.

6.3.1. Cluster Cut-Height Selection

We use a hierarchical agglomerative clustering algorithm [78] to cluster the websites based on the distance matrices. During HAC, a cut height parameter is required to determine the dissimilarity threshold at which clusters are allowed to be merged together. This parameter greatly influences the clustering accuracy, as measured by the Rand index, of the final clusters produced. For instance, using a very high cut height or dissimilarity threshold would result in most websites being included in one giant cluster since a weak measure of similarity is enforced during the merging process.

Traditionally, a static cut height is selected based on the type of data being clustered. Because website input attributes can have very different similarities and still be

related, we deploy two methods for automatically selecting the optimal cut heights, one unsupervised and one supervised. In instances where no dependable source of ground truth data is readily available, we use a *dynamic cut height* based on the algorithm using described in [88]. While the dynamic cut height produces satisfactory results when no ground truth information is available, a better strategy is available where reliable sources of ground truth are present.

Using *optimized cut height*, the best choice is found using the Rand Index as a performance measure for each possible cut height parameter value from 0.01 to 0.99. This approach performs clustering and subsequent Rand Index scoring at all possible dendogram height cutoffs using supervised cut height training on the ground truth data. The resulting cut height selected represents the dissimilarity threshold which produces the most accurate clustering results against the ground truth data according to the Rand Index score. For example, fake-escrow website HTML tags produce clusterings with Rand Index scores ranging from 0 to 97.9% accuracy while varying only the cut height parameter. Fiigure 6.3 shows fake-escrow website HTML tags generating the highest Rand Index score of 0.979 at a cut height of 0.73 with the Rand Index score quickly descending back to 0 as the cut height is increased from 0.73 to 1.00. Other fake-escrow website input attributes such as sentences, file names, and images, produce their highest Rand Index scores at differing cut height values $(0.86, 0.67, 0.29$ respectively).

These results detailed in Section 6.4.2 demonstrate that the optimized cut height approach produces more accurate clusters than dynamic cut height selection, provided that suitable ground truth data is available to find the recommended heights. Furthermore, we also note that the optimized cut height approach to perform more consistently, selecting the same top performing input attributes during training and testing executions on both data populations.

6.3.2. Individual Clustering

Because different categories of criminal activity may betray their likenesses in different ways, we need a general process that can select the best combination of input attributes for each dataset. We cannot know, a priori, which input attributes are most informative in revealing logical copies. Hence, we start by clustering on each individual attribute independently, before combining the input attributes as described below. It is indeed quite plausible that a single attribute better identifies clusters than does a combination. The clusters are selected using the two cut-height methods outlined above.

6.3.3. Best Min Combined Clustering

While individual features can often yield highly accurate clustering results, different individual features or even different combinations of multiple features may perform better across different populations of criminal websites as our results will show. Combining multiple distance matrices into a single "merged" matrix could be useful when different input attributes are important.

However, combining orthogonal distance measures into a single measure must necessarily be an information-lossy operation. A number of other consensus clustering methods have been proposed [4, 43, 57, 64], yet as we will demonstrate in the next section, these algorithms do not perform well when linking together replicated scam websites, often yielding less accurate results than clusterings based on individual input attributes.

Figure 6.2: Examples of replicated website content and file structures for the HYIP dataset.

Consequently, we have developed a simple and, in practice, more accurate approach to combining the different distance matrices. We define the pairwise distance between two websites a and b as the *minimum* distance across all input attributes. The rationale for doing so is that a website may be very different across one measure but similar according to another. Suppose a criminal manages to change the textual content of many sentences on a website, but uses the same underlying HTML code and file structure. Using the minimum distance ensures that these two websites are viewed as similar. Figure 6.2 demonstrates examples of both replicated website content and file structures. The highlighted text and file structures for each website displayed are nearly identical. One could also imagine circumstances in which the average or maximum distance among input attributes was more appropriate. We calculate those measures too, but found that the minimum approach worked best and so only those results are reported.

We created combined distance matrices for all possible combinations of distance matrices. In the case of the four input attributes considered in this paper, that means

we produced eleven combined matrices (sentences and DOM tags, sentences and file structures, sentences and images, DOM tags and file structures, DOM tags and images, file structure and images, sentences and DOM tags an file structure, sentences and DOM tags and images, sentences and file structures and images, DOM tags and file structures and images, plus sentences and DOM tags and file structures and images). In situations where many additional features are used, several specifically targeted feature combinations could also be identified for creating a limited number of combined distance matrices.

Combined clusterings are computed for each combined distance matrix using both cut-height selection methods. Ultimately, the top performing individual attribute or combination is selected based on the accuracy observed when evaluating the labeled training data set.

6.4. Evaluation Against Ground-Truth Data

One of the fundamental challenges to clustering logical copies of criminal websites is the lack of ground-truth data for evaluating the accuracy of automated methods. Some researchers have relied on expert judgment to assess similarity, but most forego any systematic evaluation due to a lack of ground truth (e.g., [91]). We now describe a method for constructing ground truth datasets for samples of fake-escrow services and high-yield investment programs.

We developed a software tool to expedite the evaluation process. This tool enabled pairwise comparison of website screenshots and input attributes (i.e., website text sentences, HTML tag sequences and file structure) by an evaluator.

	Fake-Escrow Services			
	Dynamic Cut Height		**Optimized Cut Height**	
	Train	Test	Train	Test
Sentences	0.107	0.289	**0.982**	**0.924**
Dom Tags	0.678	**0.648**	0.979	0.919
File Names	0.094	0.235	0.972	0.869
Images	0.068	0.206	0.325	0.314
S & D	**0.942**	0.584	**0.982**	**0.925**
S & F	0.120	0.245	0.980	0.895
S & I	0.072	0.257	0.962	0.564
D & F	0.558	0.561	0.979	0.892
D & I	0.652	0.614	0.599	0.385
F & I	0.100	0.224	0.518	0.510
S & D & F	0.913	0.561	0.980	0.895
S & D & I	0.883	0.536	0.971	0.673
S & F & I	0.100	0.214	0.975	0.892
D & F & I	0.642	0.536	0.831	0.772
S & D & F & I	**0.941**	0.536	0.971	0.683
	High-Yield Investment Programs			
	Dynamic Cut Height		**Optimized Cut Height**	
	Train	Test	Train	Test
Sentences	**0.713**	**0.650**	**0.738**	**0.867**
Dom Tags	0.381	0.399	0.512	0.580
File Names	0.261	0.299	0.254	0.337
Images	0.289	0.354	0.434	0.471
S & D	0.393	0.369	0.600	0.671
S & F	0.291	0.310	0.266	0.344
S & I	0.290	0.362	0.437	0.471
D & F	0.309	0.358	0.314	0.326
D & I	0.302	0.340	0.456	0.510
F & I	0.296	0.289	0.397	0.336
S & D & F	0.333	0.362	0.319	0.326
S & D & I	0.319	0.350	0.459	0.510
S & F & I	0.303	0.289	0.398	0.336
D & F & I	0.320	0.337	0.404	0.405
S & D & F & I	0.320	0.337	0.404	0.405

(a) Adjusted Rand index for different clusterings, varying the number of input attributes considered.

	Escrow	HYIPs
Minimum	0.683	0.405
Average	0.075	0.443
Max	0.080	0.623
Best Min.	**0.985**	**0.867**
DISTATIS	0.070	0.563
Clue SE	0.128	0.245
Clue DWH	0.126	0.472
Clue GV3	0.562	0.508
Clue soft/symdiff	0.095	0.401
Click trajectories [95]	0.022	0.038

(b) Adjusted Rand index for different clusterings.

Table 6.1: Table evaluating various consensus and combined clustering methods against ground truth dataset.

6.4.1. Performing Manual Ground Truth Clusterings

After the individual clusterings were calculated for each input attribute, websites could be sorted to identify manual clustering candidates which were placed in the exact same clusters for each individual input attribute's automated clustering. Populations of websites placed into the same clusters for all four input attributes were used as a starting point in the identification of the manual ground truth clusterings.

These websites were then analyzed using the comparison tool in order to make a final assessment of whether the website belonged to a cluster. Multiple passes through the website populations were performed in order to place them into the correct manual ground truth clusters. When websites were identified which did not belong in their original assigned cluster, these sites were placed into the unassigned website population for further review and other potential clustering opportunities.

Deciding when to group together similar websites into the same cluster is inherently subjective. We adopted a broad definition of similarity, in which sites were grouped together if they shared most, but not all of their input attributes in common. Furthermore, the similarity threshold only had to be met for one input attribute. For instance, HYIP websites are typically quite verbose. Many such websites contain 3 or 4 identical paragraphs of text, along with perhaps one or two additional paragraphs of completely unique text. For the ground-truth evaluation, we deemed such websites to be in the same cluster. Likewise, fake-escrow service websites might appear visually identical in basic structure for most of the site. However, a few of the websites assigned to the same cluster might contain extra web pages not present in the others.

We note that while our approach does rely on individual input attribute clusterings as a starting point for evaluation, we do not consider the final combined clustering in the evaluation. This is to maintain a degree of detachment from the combined clustering method ultimately used on the datasets. We believe the manual clusterings identify a majority of clusters with greater than two members. Although the manual clusterings contain some clusters including only two members, manual clustering efforts were ended when no more clusters of greater than two members were being identified.

6.4.2. Results

In total, we manually clustered 687 of the 4 188 HYIP websites and 684 of the 1 220 fake-escrow websites. The manually clustered websites were sorted by the date each website was identified, and then both datasets were divided into training and testing populations of 80% and 20% respectively. The test datasets represented 20% of the most recent websites identified within both the fake-escrow services and HYIP datasets. Both datasets were divided in this manner to effectively simulate the optimized combined clustering algorithm's performance in a real world setting.

In such a scenario, ground truth data would be collected for some period of time and used as training data. Once the training dataset was complete, Rand Index optimized cut heights and top performing individual or combined input attributes would be selected using the training data. Going forward, the optimized cut heights would be used during optimized combined clustering to cluster all new websites identified using the top performing individual or combined input attribute matrices. Chronologically splitting the training and test data in this manner is consistent with how we expect operators fighting cybercrime to use the method.

We computed an adjusted Rand index [130] to evaluate the combined clustering method described in Section 6.3 against the constructed ground-truth datasets using an optimized cut height which was determined from the training datasets. The optimized cut height was identified by empirically testing cut height values between 0.01 and 0.99 in increments of 0.01 against the training data. Figure 6.3 illustrates the Rand index values by input attribute at each of these intervals. The optimized Rand index value selected is indicated by the dotted line on each input attribute's chart. Finally, the cut heights selected during the training phase are used to perform optimized combined clustering against the testing data to assess how this technique might perform in the real world setting previously described above. We also evaluated

employing the unsupervised dynamic tree cut using the method described in [88] to determine an appropriate cut height along with other consensus clustering methods for comparison. Rand index scores range from 0 to 1, where a score of 1 indicates a perfect match between distinct clusterings.

Table 6.1a shows the adjusted Rand index for both datasets and all combinations of input attributes using the dynamic and optimized cut height combined clustering methods. The first four rows show the Rand index for each individual clustering. For instance, for fake-escrow services, clustering based on HTML tags alone using a dynamically determined cut height yielded a Rand index of 0.678 for the training population. Thus, clustering based on tags alone is much more accurate than by website sentences, file structure, or image similarity alone (Rand indices of 0.107, 0.094, and 0.068 respectively). When combining these input attributes, however, we see further improvement. Clustering based on taking the minimum distance between websites according to HTML tags and sentences yield a Rand index of 0.942, while taking the minimum of all input attributes yields an adjusted Rand index of 0.941. Both combined scores far exceed the Rand indices for any of the other individual input attributes using a dynamically determined cut height.

Results on the test population, for fake-escrow services, show that using the dynamic cut height method may not always produce consistent performance results. While the combined matrices achieve the highest Rand index during training, individual HTML tags outperformed all other input attributes by a large margin at 0.648 in the test population.

The optimized cut height algorithm, however, consistently demonstrates a more stable performance selecting the individual sentences matrix and the combined sentences and HTML tags matrix as the top performers in both the training and test populations.

Because cybercriminals act differently when creating logical copies of website for different types of scams, the input attributes that are most similar can change. For example, for HYIPs, we can see that clustering by website sentences yields the most accurate individual Rand index, instead of HTML tags as is the case for fake-escrow services. We can also see that for some scams, combining input attributes does not yield a more accurate clustering. Clustering based on the minimum distance of all four attributes yields a Rand index of 0.405 on the optimized cut height's test population, far worse than clustering based on website sentences alone. This underscores the importance of evaluating the individual distance scores against the combined scores, since in some circumstances an individual input attribute or a combination of a subset of the attributes may fare better.

However, it is important to point out that the optimized cut height algorithm appears to more consistently select top performing input matrices and higher Rand index scores on all of the data we benchmarked against. Rand index scores dropped in both the fake-escrow services and HYIP test datasets using a dynamically determined cut height (0.294 and 0.63 respectively). When using optimized combined clustering, however, this decrease was smaller in the fake-escrow services test population at 0.057 while test results for the HYIP data actually improved from 0.738 to 0.867 for an increase of 0.129 or 12.9%.

We used several general-purpose consensus clustering methods from R's Clue package [69] as benchmarks against the our "best min optimized cut-height" approach:

1. **"SE"** - Implements "a fixed-point algorithm for obtaining soft least squares Euclidean consensus partitions " by minimizing using Euclidean dissimilarity [43, 69].

2. **"DWH"** - Uses an extension of the greedy algorithm to implement soft least squares Euclidean consensus partitions [43, 69].

3. **"GV3"** - Utilizes an SUMT algorithm which is equivalent to finding the membership matrix m for which the sum of the squared differences between $C(m) = mm'$ and the weighted average co-membership matrix $\sum_b w_b C(_m b)$ of the partitions is minimal [64, 69].

4. **"soft/symdiff"** - Given a maximal number of classes, uses an SUMT approach to minimize using Manhattan dissimilarity of the co-membership matrices coinciding with symdiff partition dissimilarity in the case of hard partitions [57, 69].

Table 6.1b summarizes the best-performing measures for the different combined and consensus clustering approaches. We can see that our "best min optimized cut-height" approach performs best. It yields more accurate results than other general-purpose consensus clustering methods, as well as the custom clustering method used to group spam-advertised websites by the authors of [95].

6.5. Examining the Clustered Criminal Websites

We now apply the dynamic cut-height clustering methods presented earlier to the entire fake-escrow (considering sentences, DOM tags and file structure) and HYIP datasets (considering sentences alone). The 4 191 HYIP websites formed 864 clusters of at least size two, plus an additional 530 singletons. The 1 216 fake-escrow websites observed between January and June 2013 formed 161 clusters of at least size two, plus seven singletons.

6.5.1. Evaluating Cluster Size

We first study the distribution of cluster size in the two datasets. Figure 6.4a plots a CDF of the cluster size (note the logarithmic scale on the x-axis). We can see from the blue dashed line that the HYIPs tend to have smaller clusters. In addition to the 530 singletons (40% of the total clusters), 662 clusters (47% of the total) include

between 2 and 5 websites. 175 clusters (13%) are sized between 6 and 10 websites, with 27 clusters including more than 10 websites. The biggest cluster included 20 HYIP websites. These results indicate that duplication in HYIPs, while frequent, does not occur on the same scale as many other forms of cybercrime.

There is more overt copying in the escrow-fraud dataset. Only 7 of the 1 216 escrow websites could not be clustered with another website. 80 clusters (28% of the total) include between 2 and 5 websites, but another 79 clusters are sized between 6 and 20. Furthermore, two large clusters (including 113 and 109 websites respectively) can be found. We conclude that duplication is used more often as a criminal tactic in the fake-escrow websites than for the HYIPs.

Another way to look at the distribution of cluster sizes is to examine the rank-order plot in Figure 6.4b. Again, we can observe differences in the structure of the two datasets. Rank-order plots sort the clusters by size and show the percentage of websites that are covered by the smallest number of clusters. For instance, we can see from the red solid line the effect of the two large clusters in the escrow-fraud dataset. These two clusters account for nearly 20% of the total escrow-fraud websites. After that, the next-biggest clusters make a much smaller contribution in identifying more websites. Nonetheless, the incremental contributions of the HYIP clusters (shown in the dashed blue line) are also quite small. This relative dispersion of clusters differs from the concentration found in other cybercrime datasets where there is large-scale replication of content.

6.5.2. Evaluating Cluster Persistence

We now study how frequently the replicated criminal websites are re-used over time. One strategy available to criminals is to create multiple copies of the website in parallel, thereby reaching more victims more quickly. The alternative is to re-use copies in a serial fashion, introducing new copies only after time has passed or the

prior instances have collapsed. We investigate both datasets to empirically answer the question of which strategy is preferred.

Figure 6.5 groups the 10 largest clusters from the fake-escrow dataset and plots the date at which each website in the cluster first appears. We can see that for the two largest clusters there are spikes where multiple website copies are spawned on the same day. For the smaller clusters, however, we see that websites are introduced sequentially. Moreover, for all of the biggest clusters, new copies are introduced throughout the observation period. From this we can conclude that criminals are likely to use the same template repeatedly until stopped.

Next, we examine the observed persistence of the clusters. We define the "lifetime" of a cluster as the difference in days between the first and last appearance of a website in the cluster. For instance, the first-reported website in one cluster of 18 fake-escrow websites appeared on February 2, 2013, while the last occurred on May 7, 2013. Hence, the lifetime of the cluster is 92 days. Longer-lived clusters indicate that cybercriminals can create website copies for long periods of time with impunity.

We use a survival probability plot to examine the distribution of cluster lifetimes. A survival function $S(t)$ measures the probability that a cluster's lifetime is greater than time t. Survival analysis takes into account "censored" data points, i.e., when the final website in the cluster is reported near the end of the study. We deem any cluster with a website reported within 14 days of the end of data collection to be censored. We use the Kaplan-Meier estimator [83] to calculate a survival function.

Figure 6.6 gives the survival plots for both datasets (solid lines indicate the survival probability, while dashed lines indicate 95% confidence intervals). In the left graph, we can see that around 75% of fake-escrow clusters persist for at least 60 days, and that the median lifetime is 90 days. Note that around 25% of clusters remained active at the end of the 150-day measurement period, so we cannot reason about how long

these most-persistent clusters will remain.

Because we tracked HYIPs for a much longer period (Figure 6.6 (right)), nearly all clusters eventually ceased to be replicated. Consequently, the survival probability for even long-lived clusters can be evaluated. 20% of HYIP clusters persist for more than 500 days, while 25% do not last longer than 100 days. The median lifetime of HYIP clusters is around 250 days. The relatively long persistence of many HYIP clusters should give law enforcement some encouragement: because the criminals reuse content over long periods, tracking them down becomes a more realistic proposition.

6.6. Related Work

A number of researchers have applied machine learning methods to cluster websites created by cybercriminals. Wardman et al. examined the file structure and content of suspected phishing webpages to automatically classify reported URLs as phishing [152]. Layton et al. cluster phishing webpages together using a combination of k-means and agglomerative clustering [89].

Several researchers have classified and clustered web spam pages. Urvoy et al. use HTML structure to classify web pages, and they develop a clustering method using locality-sensitive hashing to cluster similar spam pages together [144]. Lin uses HTML tag multisets to classify cloaked webpages [99]. Lin's technique is used by Wang et al. [149] to detect when the cached HTML is very different from what is presented to user. Finally, Anderson et al. use image shingling to cluster screenshots of websites advertised in email spam [13]. Similarly, Levchenko et al. use a custom clustering heuristic method to group similar spam-advertised web pages [95]. We implemented and evaluated this clustering method on the cybercrime datasets in Section 6.4. Der et al. clustered storefronts selling counterfeit goods by the affiliate structure driving traffic to different stores [41]. Finally, Leontiadis et al. group similar unlicensed

online pharmacy inventories [91]. They did not attempt to evaluate against ground truth; instead they used Jaccard distance and agglomerative clustering to find suitable clusters.

Neisius and Clayton also studied high-yield investment programs [117]. Notably, they estimated that a majority of HYIP websites used templates licensed from a company called "Goldcoders". While we did observe some Goldcoder templates in our own datasets, we did not find them occurring at the same frequency. Furthermore, our clustering method tended to link HYIP websites more by the rendered text on the page rather than the website file structure.

Separate to the work on cybercriminal datasets, other researchers have proposed consensus clustering methods for different applications. DISTATIS is an adaptation of the STATIS methodology specifically used for the purposes of integrating distance matrices for different input attributes [5]. DISTATIS can be considered a three-way extension of metric multidimensional scaling [87], which transforms a collection of distance matrices into cross-product matrices used in the cross-product approach to STATIS. Consensus can be performed between two or more distance matrices by using DISTATIS and then converting the cross-product matrix output into into a (squared) Euclidean distance matrix which is the inverse transformation of metric multidimensional scaling [3].

Our work follows in the line of both of the above research thrusts. It differs in that it considers multiple attributes that an attacker may change (site content, HTML structure and file structure), even when she may not modify all attributes. It is also tolerant of greater changes by the cybercriminal than previous approaches. At the same time, though, it is more specific than general consensus clustering methods, which enables the method to achieve higher accuracy in cluster labelings.

6.7. Concluding Remarks

When designing scams, cybercriminals face trade-offs between scale and victim susceptibility, and between scale and evasiveness from law enforcement. Large-scale scams cast a wider net, but this comes at the expense of lower victim yield and faster defender response. Highly targeted attacks are much more likely to work, but they are more expensive to craft. Some frauds lie in the middle, where the criminals replicate scams but not without taking care to give the appearance that each attack is distinct.

In this paper, we propose and evaluate a combined clustering method to automatically link together such semi-automated scams. We have shown it to be more accurate than general-purpose consensus clustering approaches, as well as approaches designed for large-scale scams such as phishing that use more extensive copying of content. In particular, we applied the method to two classes of scams: HYIPs and fake-escrow websites.

The method could prove valuable to law enforcement, as it helps tackle cybercrimes that individually are too minor to investigate but collectively may cross a threshold of significance. For instance, our method identifies two distinct clusters of more than 100 fake escrow websites each. Furthermore, our method could substantially reduce the workload for investigators as they prioritize which criminals to investigate.

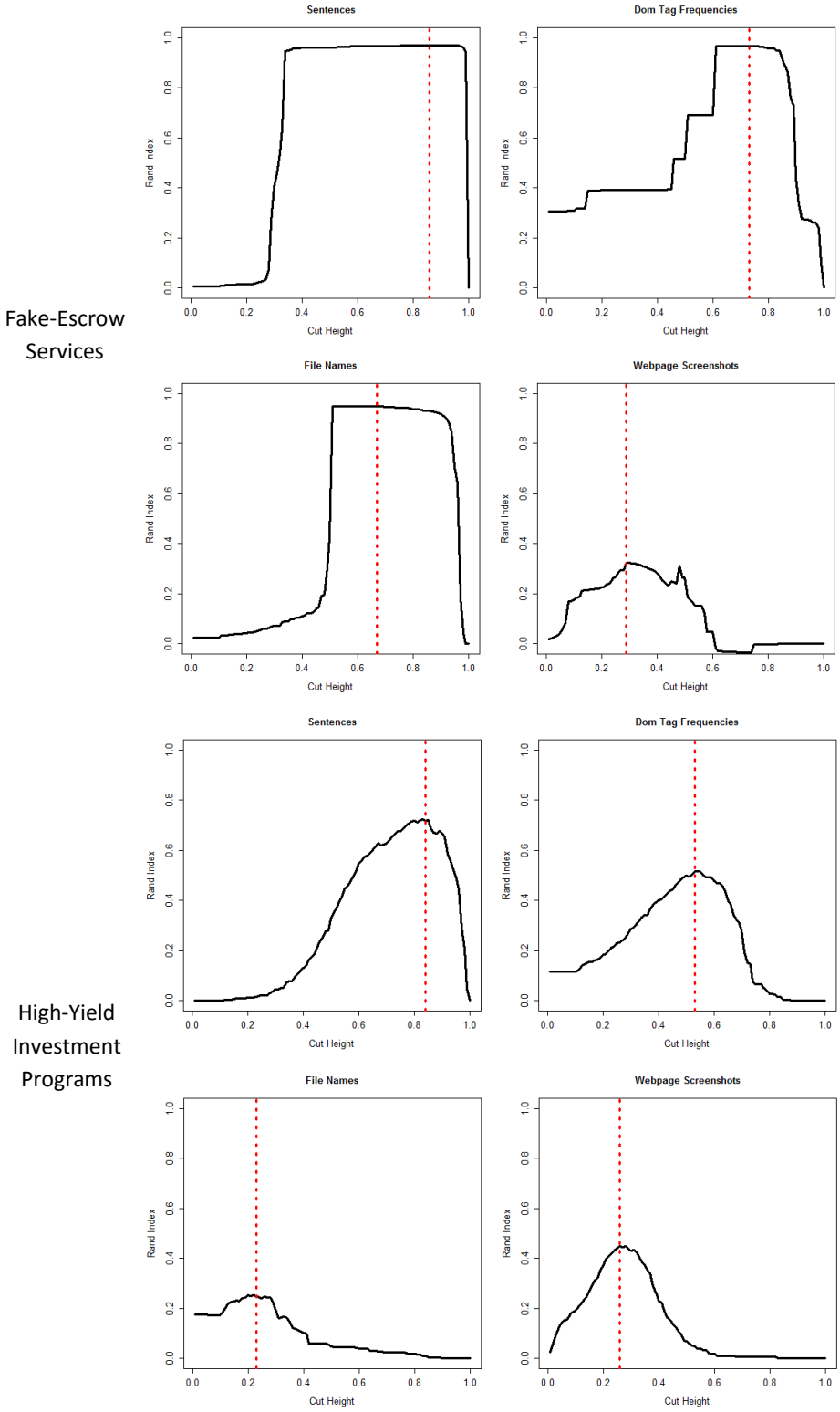
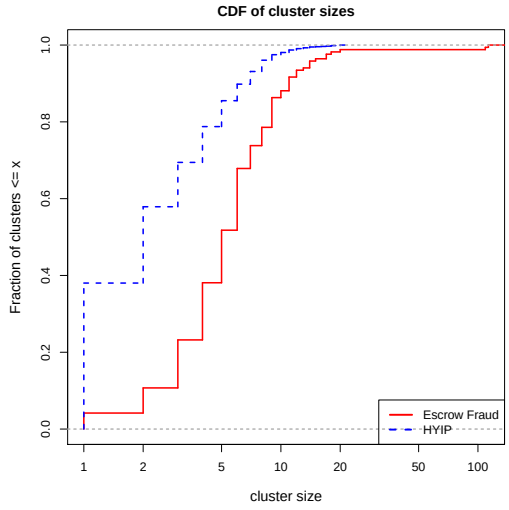
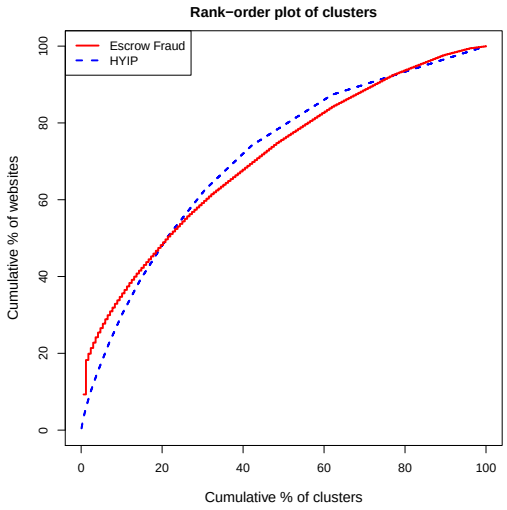


Figure 6.3: Rand index values for each input attribute at various cut heights.



(a) Cumulative distribution function of cluster size.



(b) Rank order plot of cluster sizes.

Figure 6.4: Evaluating the distribution of cluster size in the escrow fraud and HYIP datasets.

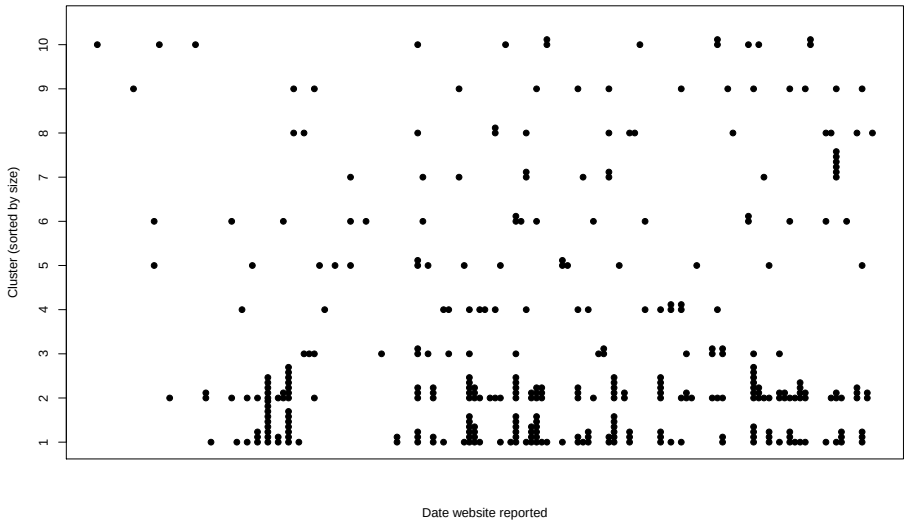


Figure 6.5: Top 10 largest clusters in the fake-escrow dataset by date the websites are identified.

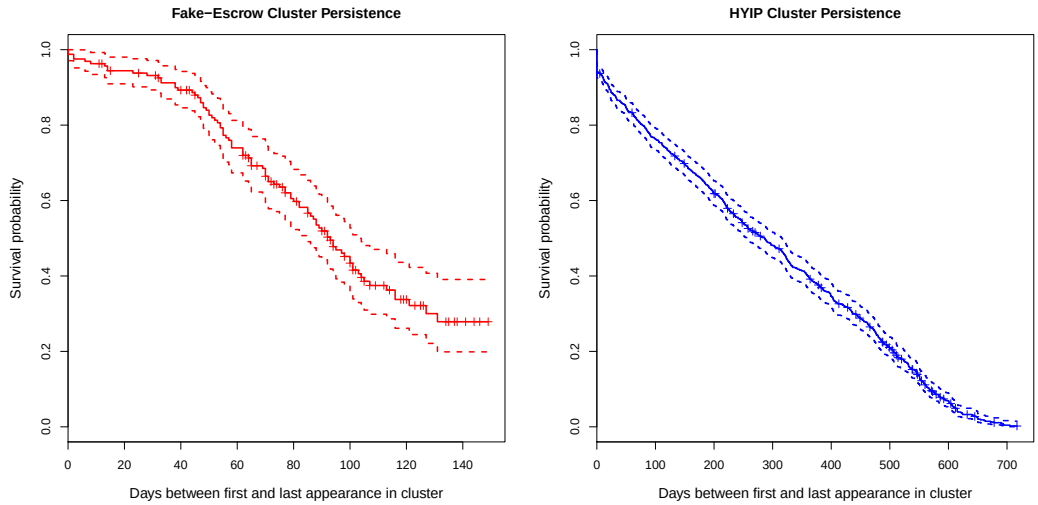


Figure 6.6: Survival probability of fake-escrow clusters (left) and HYIP clusters (right).

Chapter 7

DETECTION OF WEBSITES SELLING COUNTERFEIT GOODS

7.1. Introduction

The economist George Akerlof won a Nobel Prize for his work describing markets with asymmetric information [9]. In such a market for "lemons", consumers cannot reliably distinguish between high- and low-quality goods. This inability triggers a crucial market failure, whereby prices are driven down and low-quality goods dominate.

More recently, Anderson argued that the market for secure software is also a lemons market, as buyers cannot reliably observe whether or not the software they are buying is in fact secure [14]. Researchers working in security economics now recognize that information asymmetry is one of the fundamental barriers facing cybersecurity today [15].

In this chapter, we study a related market for lemons: the online sale of clothing and luxury goods. Miscreants selling knockoff versions of popular goods have proliferated online in recent years. Frequently, counterfeit-goods shops are manipulating search engines to get high placement in search results for legitimate terms, thereby duping consumers into thinking they can get a good deal on the real thing. If search engines cannot distinguish legitimate sellers from those selling counterfeits, why would we expect untrained people to be capable of doing so?

The online sale of knockoffs matters, and not just for those brands whose goods are impersonated. Bad experiences with e-commerce carry large indirect costs, in that

they can turn people off from online participation and erode trust in the Internet [132]. Consequently, in this paper we set out to investigate the prevalence of counterfeit goods online. We make the following contributions.

- We build an accurate classifier using features automatically extracted from website content that distinguishes legitimate from fake sellers based upon data returned by search results (Section 7.2).
- We apply the trained classifier and find that 5 407 websites are selling counterfeits, out of a total of 18 756 websites. Overall, 32% of search results point to fakes, and 79% of queries issued included at least one fake in the first page of results.
- We show that while search engines do refer customers to counterfeit sellers when the search terms ask for it, they also refer customers to counterfeit sellers in large numbers even when they give no indication that they want fakes.
- We present a linear regression that demonstrates that the higher the selling price is for fakes for a given brand, the more search results point to fakes. The regression also demonstrates that active enforcement (as measured by DMCA takedowns) can reduce the prevalence of fakes in search results by 9 %-pts.
- We show that the prevalence of fakes among brands is relatively stable over time, and furthermore that some sellers respond to their website dropping out of search results by adding copies registered at different URLs.

7.2. Classifying Websites Selling Counterfeits

We first describe the features used in building the classifier in Section 5.2.4 and then outline the methods used in Section 7.2.1.

	GLM		SVM		ADA	
	#	%	#	%	#	%
TP	175	29.1%	180	29.9%	125	20.8%
TN	337	66.0%	340	56.5%	318	52.8%
FP	31	5.1%	28	4.7%	50	8.3%
FN	59	9.8%	54	9.0 %	109	18.1%
Accuracy	85.0%		86.4%		73.6%	
Precision	85.0%		86.5%		71.4%	
Recall	74.8%		76.9%		54.4%	

Table 7.1: Truth tables and accuracy measures for each classifier using 10-fold cross-validation.

7.2.1. Building and evaluating the classifiers

Counterfeit websites were classified using Logistic Regression, Adaptive Boosting, and Support Vector Machine models. Each of the machine learning models were trained against our specialized features developed to identify counterfeit goods websites. Finally, the models were validated against a manually constructed ground truth dataset to assess each model's detailed classification accuracy characteristics.

We implemented the three models using the R programming language. Three of these packages include Support Vector Machines (SVM) [107], Generalized Linear Models (GLM) [133], and Adaptive Boosting (AdaBoost) [36]. While all three packages can be highly accurate for various types of classification problems, each package performs very differently when modeling (i.e. learning) different volumes of input data [51].

Ground Truth Identification One of the fundamental challenges to classifying logical copies of counterfeit goods websites is the lack of ground-truth data available for evaluating the accuracy of automated feature selection and classification methods.

Some researchers have relied on expert judgment to assess similarity, (e.g., [91]) but most forego any systematic evaluation due to a lack of ground truth. We now describe a method for constructing ground truth datasets for samples of counterfeit goods websites.

In order to classify stores, 602 unique screenshot/HTML pairs were pulled at random from the data collected for manual inspection. The screenshots were then examined to determine whether the store appeared to be selling fakes – a task which is easier, and much slower, to do by hand. We relied on the judgment of a human reviewer who had viewed many manually curated examples of legitimate and counterfeit websites. Of the 602 stores sampled, 234 were determined to be counterfeit and 368 were determined to not be counterfeit.

Results We independently trained and evaluated logistic regression (GLM), Support Vector Machine (SVM) and, adaptive boosting (ADA) models using 10-fold cross-validation. Table 7.1 shows the detailed truth tables for each model, along with figures for accuracy, precision and recall. Logistic regression and SVM produced more accurate results than adaptive boosting.

To get a sense of the relative importance of different features in the classifier, we can examine the coefficients and odds ratios from the best-fit logistic regression trained on the ground-truth data. Table 7.2 presents the results, with terms that are statistically significant in the model highlighted in bold. As expected, the presence of a large IFrame loading an external website is highly associated with the website selling counterfeits. In fact, websites exhibiting this behavior face 204-times greater odds that they are selling counterfeits! Newly registered domains, using the term 'replica' in the FQDN, and appearing on a takedown page are all associated with selling fakes. The more currencies available on a website, the greater the advertised savings and the longer the FQDN, the more likely the website is to be fake. Surprisingly, however,

Feature	Coef.	Odds ratio	p-value
Page Contains Webmail Address	0.697	2.007	0.1722
Unique Brand Term Count	0.167	1.182	< 0.0001
# Currencies Seen	0.240	1.272	0.0017
Large IFrames	5.320	204.3	< 0.0001
Private or China WHOIS	0.285	1.330	0.384021
Replica in FQDN	1.442	4.227	0.0002
WHOIS Registration < 1 Year	1.505	4.504	0.0001
Percent Savings Average	0.044	1.045	< 0.0001
# Times Duplicate Price Seen	0.005	1.005	0.4471
Top-Level Page Mentions Brand	-0.701	0.496	0.0097
Website on Takedown Page	2.892	18.05	0.0005
Length of FQDN	0.044	1.045	0.0782
Website in Alexa Top 100K	-2.626	0.072	< 0.0001

Table 7.2: Coefficients and odds ratios for the logistic regression classifier (terms in bold are statistically significant).

a private or Chinese WHOIS registration address was not found to be statistically significant. Finally, two features are negatively associated with selling fakes – websites with a top 100,000 Alexa ranking and those whose top-level index page also mention the brand are less likely to sell fakes. The latter reflects the fact that the website is more likely to be an actual merchant and not a compromised host.

7.2.2. The Blended Model Approach

We created a high recall blended model using 8 bootstraps each for the Adaptive Boosting, Support Vector Machines, and linear regression libraries. The 8 bootstrap training samples were created by taking 8 random samples of 20% with replacement from the 80% population of data reserved for training. Eight models were created for each individual machine learning library using these bootstraps.

Predictions were made using our reserved test data against all bootstrap models. Next, resulting probabilities for each prediction were ranked to identify predictions with the highest probability of belonging to the counterfeit goods class. By summing

all 8 rankings for each modeling library, an independent bootstrap consensus could then be determined for the Adaptive Boosting, Support Vector Machines, and Linear Regression libraries.

We calculated the accuracy of each independent consensus at various cut-points to determine which blended model had the highest accuracy. For instance, accuracy for each blended model was calculated on the first 150, 200, 250, 500, 1000, and 1500 predictions. Since all predictions are sorted by the probability of belonging to the counterfeit goods class, population accuracy increases when looking at only the top N predictions. Using this information, all of the blended Adaptive Boosting predictions belonging to the counterfeit goods class were chosen as a starting base for the blended model. Next, the top 200 and the top 100 predictions with the greatest probability of belonging to the counterfeit goods class were selected from SVM and GLM respectively. Any missing predictions from either of these populations were added to the blended result.

While this approach is slightly less accurate overall when compared to Adaptive Boosting (e.g. 97.8% vs. 96.3%), the blended model produces much higher recall at 100% never predicting incorrectly that a counterfeit goods website does not belong to the counterfeit goods class. Adaptive Boosting is able identify more total counterfeit goods class members at 512 vs. 472 for the blended approach. However, the blended approach is more effective at identifying when a website should not be associated with the counterfeit goods class.

7.2.3. Counterfeit Goods Classification Feature Importance

The Adaptive Boosting package for R includes a variable importance plot which can be used to determine the features which were considered most important during the model's training. These features provide a sense of variable importance using the frequency at which each feature was selected for boosting during the training

process. [36] Figure 7.1 shows a pairwise plot of each feature and the importance score assigned by the Adaptive Boosting model when training against the 80% randomly sampled training data set. This model shows both the page level features and search term specific features working together to provide the optimal prediction results produced by the Adaptive Boosting model. Page level features such as privately registered webpages, webpages originating from China, and the webpage's registration country appear to highly influence prediction of counterfeit websites. In addition, search term specific features such as unique brand counts greater than 15, and specific individual brands such as Panerai, Louis Vuitton, Bvlgari, Fendi, and Burberry make appearances as top predictors on the Model's important variables list.

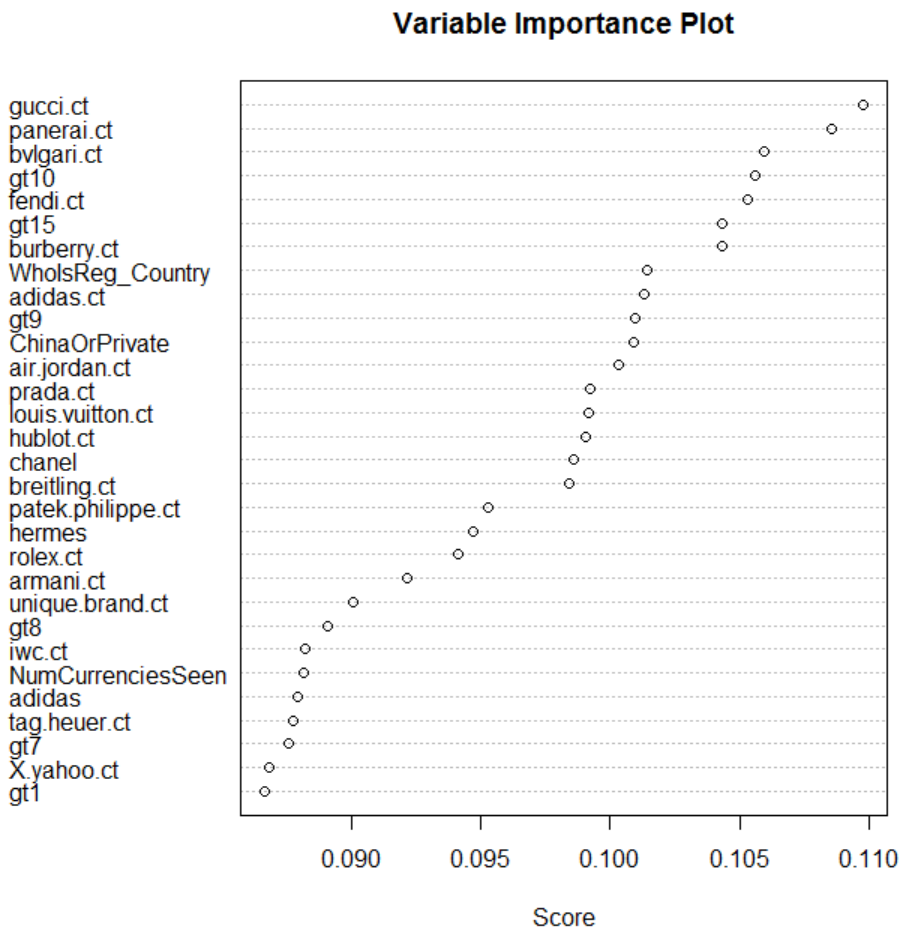


Figure 7.1: Important Features Selected By Adaptive Boosting During Model Training

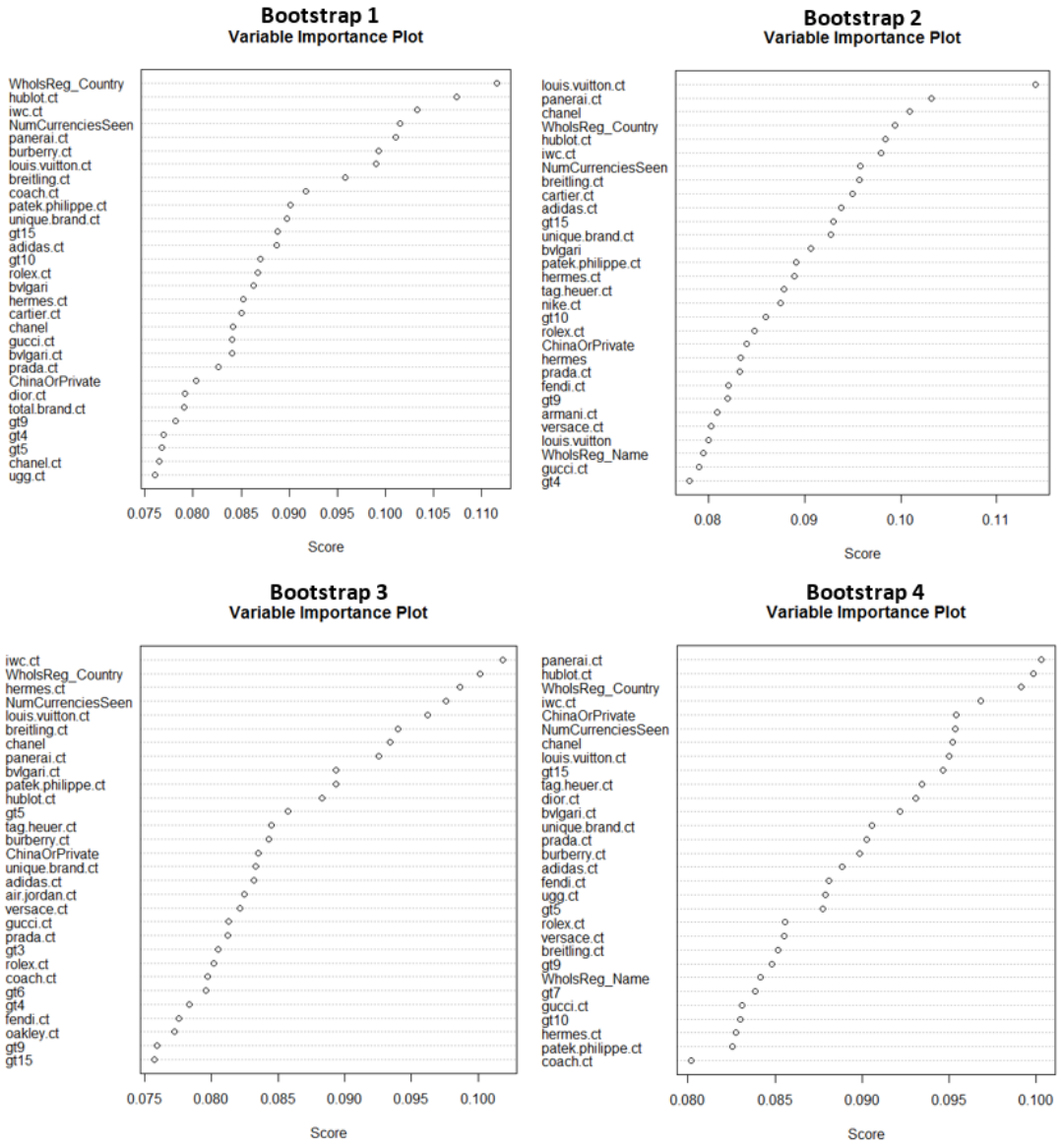


Figure 7.2: Important Features Selected By Adaptive Boosting During Model Training (Bootstraps 1-4)

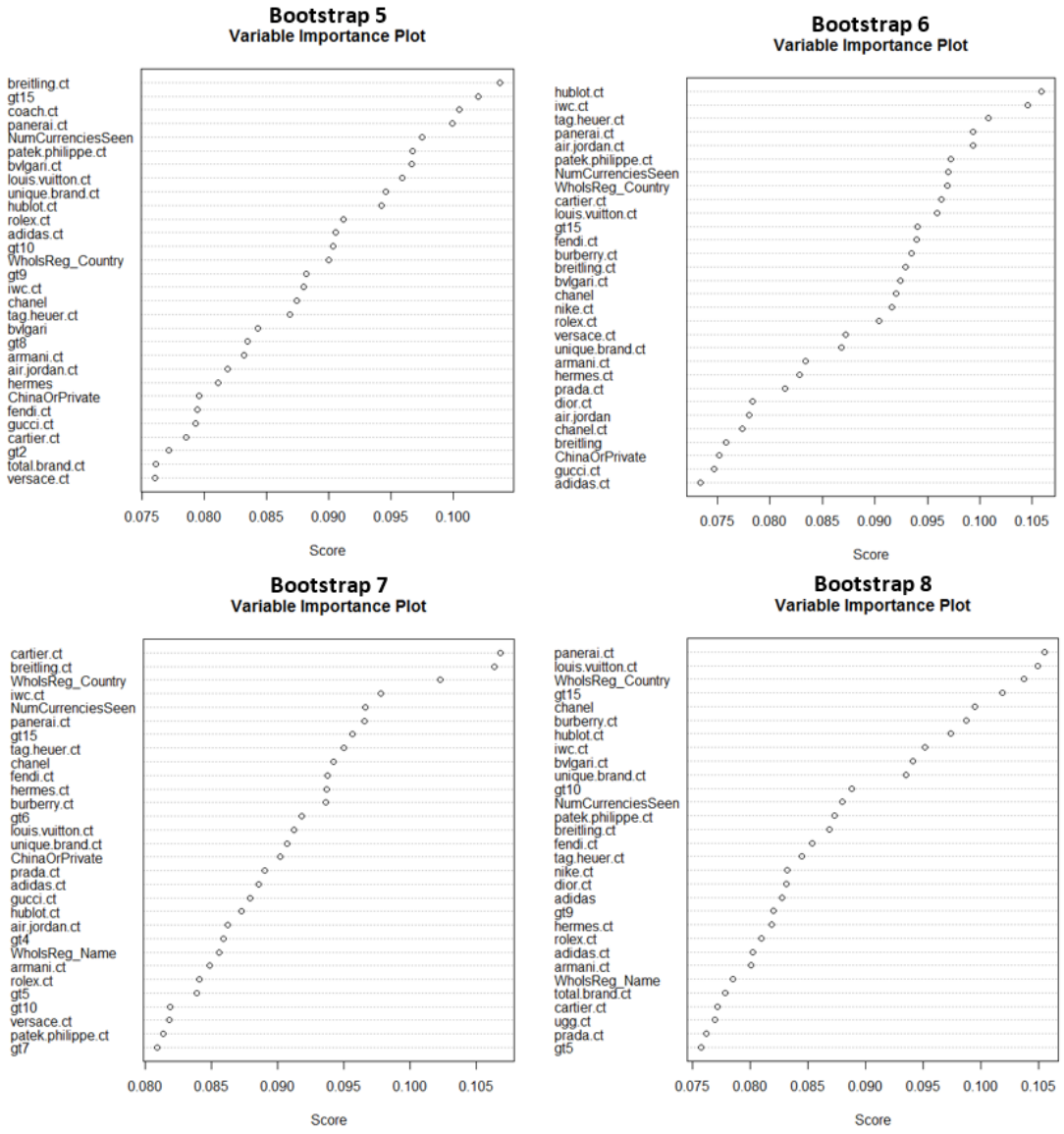


Figure 7.3: Important Features Selected By Adaptive Boosting During Model Training (Bootstraps 5-8)

While the important variables listed above are highly informative, still greater insights can be gained by looking further at the eight random bootstrap sample models created for Adaptive Boosting during the blended model creation process. In practice, looking at the average performance for variable importance has proven to be highly informative producing a better representation of each variables overall contribution

to Adaptive Boosting model predictions. [36] Figures 7.2 and 7.3 illustrates one variable importance plot for each of the eight random bootstrap samples of 20% with replacement taken from the 80% training data population.

These plots reveal that the webpage's WHOIS registration country is consistently chosen as a "top 4" feature in 6 out of 8 models. It is also chosen as a "top 10" feature in 7 out of the 8 models. In addition, The number of currencies seen on a webpage is chosen as a "top 10" feature in 7 out of the 8 models as well. Webpages offering more than 10 or 15 unique search term specific brands also appears to strongly influence accurate predictions in most models. Furthermore, specific individual brands such as Hublot, IWC, Louis Vuitton, Chanel, and Panerai are consistently identified by each model as top predictors of counterfeit goods websites.

7.3. Related Work

In very recent work carried out concurrently to our own efforts, Wang et al. use clustering techniques to identify "campaigns" of similar websites advertising counterfeit goods [148], using methods described in [42]. They also issue search queries for brands, but they identify websites selling knockoffs by looking for signs that the hosting website has been hacked and is demonstrating cloaking behavior. Their analysis in turn focuses on linking together disparate websites into groupings. Our work complements theirs in that we focus on the more general problem of classifying all search results as selling knockoffs or not. Cloaking behavior is indicative of many, but certainly not all, of today's websites selling counterfeits. One way we can see this is to note that 45% of the websites we identified as selling counterfeits also mentioned the brand on their homepage. This suggests that many of these websites are not hacked, but instead are brazenly selling fakes. Furthermore, we focus our analysis on examining differences in the prevalence of counterfeits in web search by user intent

and brand characteristics.

More broadly, a number of papers have investigated abuse in search-engine results. Provos et al. presented a mechanism for identifying drive-by-downloads in web search results [127]. Moore et al. [115] and John et al. [77] report on the poisoning of trending search terms to distribute malware and host ad-laden, auto-generated content. Leontiadis et al. document search-poisoning by those peddling counterfeit pharmaceuticals [90]. The same authors recently reported on a longitudinal study of such search-engine poisoning promoting unlicensed pharmacies [92]. Notably, they compared the prevalence of search poisoning based upon the intent of the search queries, finding that both innocent and complicit queries turn up unlicensed pharmacies. This complements our own findings regarding the presence of knockoffs in search results, regardless of query intent.

A number of papers have proposed classifiers to identify malicious web content. Abu Nimeh et al. compare several methods for classifying phishing websites [7]. Many others have constructed features for classifying malicious web pages based upon website content or behavior [18,27,118,147,153]. Our paper continues in this tradition, but builds a classifier based upon features specific to websites selling knockoffs (e.g., selling in multiple currencies, pricing information).

7.4. Conclusion

The web has revolutionized commerce, giving consumers access to more choice at lower prices. Unfortunately, it can be hard to determine whether the great deal found online is truly a bargain or is actually cheap because the merchant is selling knockoffs. In this paper, we have conducted a large-scale empirical analysis of 25 counterfeit goods found through web search. We designed a purpose-built classifier to predict whether a given website found through search likely sells genuine merchandise

or counterfeit goods.

We have found that 32% of inspected search results point to fakes overall, but we have also observed wide variation. Innocent queries such as "hublot buy online" are less likely to lead to fakes, but introducing the word "cheap" can lead to nearly 40% of the results pointing to stores selling counterfeits. Furthermore, some brands are targeted more often than others. Brands who sell high-end goods such as luxury watches tend to have their search results polluted with more knockoffs. Not all the news is bad for brands, however. We have presented a linear regression that indicates those who actively protect their brand (which we observe by a record of DMCA enforcement) experience much lower rates of fakes in search.

By and large, merchants selling fakes take advantage of reliable web hosting by operating in countries with strong infrastructure. They also tend to replace removed websites with copied content on new URLs.

In future work, we hope to continue measuring progress in combating the sale of counterfeit goods by carrying out longitudinal studies. More work can be done to improve the classifier's accuracy so that it can be used in an ongoing basis by operators in the field. We also hope to investigate similarities between websites selling fakes in greater depth.

Chapter 8

APPLYING STRAND TO MALWARE CLASSIFICATION

This chapter utilizes both the bioinformatics features developed in Chapter 3 and the Strand gene sequence classification software described in Chapter 4 to successfully train and classify 9 different types of malware files introduced in the Kaggle Microsoft Malware Classification Challenge (BIG 2015) [79]. I tie together the domains of cybercrime and bioinformatics building upon both the cybercrime feature extraction and detection techniques described in Chapters 5, 6, and 7. The Collaborative Analytics Framework introduced in Chapter 2 facilitates strategic and efficient changes within the Strand application to support processing the content within any number of malware input files as if they contained gene sequence data.

The Kaggle challenge simulates the file input data processed by Microsoft's real-time detection anti-malware products which are installed on over 160M computers and inspect over 700M computers each month [79]. Similar to the mutations found in gene sequences, criminals producing malware introduce polymorphism [79] into the programs which they create. Miscreants avoid malware detection in this manner by basically designing mutated or differing copies of the same program which still share the same functional characteristics. The goal of the Microsoft Malware Classification Challenge is to group these malware at a high level into 9 different classes of malicious programs.

8.1. The Training and Classification Input Data

Microsoft provided almost a half terabyte of training and classification input data which included:

1. **Bytes Files:** 10,868 training and 10,873 test .bytes files containing the raw hexadecimal representation of the file's binary content with the executable headers removed.

2. **Asm Files:** 10,868 training and 10,873 test .asm files containing a metadata manifest including data extracted by the IDA disassembler tool. This information includes things such as function calls, strings, assembly command sequences and more.

3. **Training Labels:** Each training and test file name is a MD5 hash of the actual program. The training labels file contains each MD5 hash and the malware class which it maps to. No training labels were provided for the test data input files.

4. **Sample Submission:** The sample submission file illustrates the valid submission format for 10,873 sample records.

5. **Data Sample:** The data sample file includes a preview of the test and training data.

8.2. Challenge Evaluation, Competitors, and Results

Kaggle challenge participants were evaluated using a multi-class logarithmic loss score. Each test file submission made required not only the predicted malware class, but the estimated probabilities for the file belonging to each of the 9 classes. Each submission record included the file hash and 9 additional comma-delimited fields each containing a value for the predicted probability that a given file belongs a particular class.

There were 377 international teams competing in the contest with $16,000 in available prize money. The winning team achieved a log loss ratio score of 0.002833228 where a value of zero represents a perfect score. The winning model produced an accuracy level greater than 99% during 10-fold cross-validation [80].

The winning team's model ensemble was highly complex using a combination of features including: byte 4 gram instruction counts, function names and derived assembly features, assembly op-code n-grams, asm file segment counts, and asm file pixel intensity [150]. Generating these features required 500GB of disk space for the original training data and an additional 200GB for engineered features [150]. While the final features used for the model required only 4GB, both feature engineering and generating the top performing model takes around 48 hours [150]. Furthermore, it takes an additional 24 hours to generate the best model ensemble which produced the winning score [150]. In short, the winning submission takes 72 hours to produce.

8.3. Applying Strand to Microsoft Malware Classification Challenge

I was able to created highly efficient and accurate classification results by utilizing the Strand application to train and generate predictions against the Microsoft Malware Data. While Strand was originally designed to process .fasta or .fna formatted gene sequence files, the Collaborative Analytics Framework accommodated for relatively minor modifications to read and process the malware files as input.

One benefit of Strand is that unlike many other sequence classifiers and k-mer counters [103,121,157], Strand uses no special encoding of sequence data. As a result, any characters supported by the Unicode character set can be treated as sequence data within Strand.

During gene sequence classification, the SNIP's of sequence data commonly provided by modern sequencers can be in either forward or reverse-complement order.

```
00401000 56 8D 44 24 08 50 8B F1 E8 1C 1B 00 00 C7 06 08
00401010 BB 42 00 8B C6 5E C2 04 00 CC CC CC CC CC CC CC
00401020 C7 01 08 BB 42 00 E9 26 1C 00 00 CC CC CC CC CC
00401030 56 8B F1 C7 06 08 BB 42 00 E8 13 1C 00 00 F6 44
00401040 24 08 01 74 09 56 E8 6C 1E 00 00 83 C4 04 8B C6
00401050 5E C2 04 00 CC CC CC CC CC CC CC CC CC CC CC CC
00401060 8B 44 24 08 8A 08 8B 54 24 04 88 0A C3 CC CC CC
00401070 8B 44 24 04 8D 50 01 8A 08 40 84 C9 75 F9 2B C2
00401080 C3 CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC
00401090 8B 44 24 10 8B 4C 24 0C 8B 54 24 08 56 8B 74 24
004010A0 08 50 51 52 56 E8 18 1E 00 00 83 C4 10 8B C6 5E
004010B0 C3 CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC
0042A800 ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ?? ??
```

Figure 8.1: Malware .bytes hex data file content.

As a result of this limitation, classification searches on sequence data must be made using each input sequence's forward and reverse-complement effectively doubling the number of classification searches required. This particular feature of Strand is gene sequence specific and was irrelevant for malware classification. After turning off the reverse-complement search and modifying the sequence file parsing routine, Strand was able train and classify against malware data with no other changes.

8.4. Developing Malware Features for Strand

All of the Malware feature engineering required to convert Malware program data into Strand sequences, fits into just a few lines of code. First, no .asm files were used to produce the score and accuracy results presented later in Table 8.1. The .asm files were eliminated since I was unable produce a higher accuracy score when using them in combination with the .bytes files. In the future, a two model ensemble could be created which classifies each malware file in Strand based on combining the individual scores from dual .bytes and .asm model predictions.

Figure 8.1 illustrates the typical content encountered within the .bytes hex data files provided by Microsoft. The first 8 characters of each line contain a line number, and the last line shows how some hex content is unavailable and displayed as "??".

```
StringBuilder currSeqText = new StringBuilder();

foreach (var line in File.ReadLines(FnaFileMap.FilePath))
{
    //Remove the carriage returns, line number, spaces, and ??
    //values from each line of hex in the .bytes file.
    currSeqText.Append(line.Substring(9).Replace(" ", string.Empty).Replace("?", string.Empty));
}
```

Figure 8.2: Strand C# code used to process malware .bytes hex data files.

Both the line numbers and "??" symbols are removed during Strand processing. The "??" symbols were present in the files provided by Microsoft for the challenge.

When reading each .bytes file, Strand uses the code shown in Figure 8.2 to convert the .bytes malware files into a Strand sequence. During processing each carriage return, space, and "?" character are removed. This produces a single string or Strand sequence containing all hex content read from the malware file. Once the malware hex data is cleaned, sequence words or k-mers are generated by Strand exactly as described in Chapters 3 and 4.

8.5. Malware Classification Results Using Strand

While Strand did not produce a winning log loss score for the Kaggle Microsoft Malware Classification Challenge (BIG 2015) [79], I was able achieve a log loss score of 0.452784 when using a 32-bit minhashing configuration with Strand. I used a word length 10 characters and 2400 minhash values within the Strand minhash signature to achieve this result. The primary benefit of using Strand was achieving an acceptable degree of accuracy within a short period of time. Training and classification times were both under 7 hours for processing 224GB of training data and 189GB of test data.

5-Fold Cross-validation Results					
Fold	Classified	Correct	Accuracy	Train Time	Classify Time
Fold 1	1087	979	90.06%	06:46:31	00:30:38
Fold 2	1087	998	91.81%	06:25:38	00:30:26
Fold 3	1087	1011	93.01%	06:35:01	00:31:50
Fold 4	1087	995	91.54%	06:34:53	00:33:53
Fold 5	1087	1004	92.36%	06:21:12	00:28:12

Table 8.1: Five-fold cross-validation results when using Strand to predict 5 folds from the Microsoft malware training data.

Table 8.1 shows five-fold cross-validation results for the first 5 folds of Malware Classification Challenge training data. Strand averaged 91.76% accuracy across the five folds predicted using only 32-bit hashing functions. When using 64-bit hashing functions, I was able to drastically reduce the log loss score produced from 0.452784 to 0.222864. While memory consumption increased slightly, there was no real degradation in training or classification performance.

The 64-bit training database takes up approximately 5GB in memory and 436MB on disk while the 32-bit version takes up approximately 3GB in memory and 255MB on disk. Due to the small size of the training database, multiple copies can be loaded into memory for multiple worker processes to take advantage of process level parallelism when classifying large volumes of data. For example, fifteen classification workers were used to process the test files provided by Microsoft.

5-Fold Cross-validation Results					
Fold	Classified	Correct	Accuracy	Train Time	Classify Time
Fold 1	1087	1053	96.87%	06:42:54	00:33:22
Fold 2	1087	1054	96.96%	05:53:21	00:31:36
Fold 3	1087	1069	98.34%	06:50:26	00:34:12
Fold 4	1087	1052	96.78%	06:32:24	00:35:00
Fold 5	1087	1065	97.98%	06:50:25	00:32:50

Table 8.2: Five-fold cross-validation results when using Strand with 64-bit hash codes to predict 5 folds from the Microsoft malware training data.

Table 8.2 shows five-fold cross-validation results for the version of Strand using 64-bit hash codes. Strand averaged 97.39% accuracy across the five folds. When using 64-bit hashing functions, I was able to drastically reduce the log loss score produced from 0.452784 to 0.222864. While memory consumption increased slightly, there was no large degradation in training or classification performance.

8.6. Conclusion

The Collaborative Analytics Framework in combination with Strand can be successfully used to predict multiple classes of malware data. While Strand did not produce a winning log loss score, we were able to achieve classification accuracy levels well over 90% while making predictions in less than 10% of the training and classification times required by the winning team. Strand's support for all Unicode characters makes it highly flexible and ideal for high performance machine learning applications using input data formats other than just the training and classification of .fna and .fasta formatted sequence data input files.

Chapter 9

CONCLUSION AND FUTURE WORK

9.1. Concluding Remarks

This thesis provides a wide variety of machine learning techniques related to big data applications in Bioinformatics and Cybercrime. The Collaborative Analytics Framework introduces a multicore MapReduce style processing pipeline, techniques, and tools which create additional massively parallel methods for machine learning. The framework is leveraged to solve multiple challenging research problems, ranging from gene sequence classification to identifying websites selling counterfeit goods. I have also presented parallel programming algorithms for feature space compression using techniques such as minhashing, which is a form of locality sensitive hashing applied to both supervised and unsupervised machine learning techniques. It is the multiple scalable machine learning applications in the fields of bioinformatics and cybercrime that forms the primary contribution of this thesis.

Chapter 2, applies highly parallel producer-consumer data processing pipelines to very large volumes of input data creating a novel feature extraction framework. While these techniques are similar to MapReduce style processing, they allow both the 'map' and 'reduce' stages access to the same shared memory for enhanced parallelism. Parallel extraction of features from a variety of unstructured data sources such as gene sequences, text, webpages, html, and images are discussed.

Chapter 3 expands upon the new framework introduced in Chapter 2 and presents feature extraction techniques for gene sequence data. The Collaborative Analyt-

ics Framework successfully extracts gene sequence words from unstructured gene sequence data in a format which Edit Distance is approximated by using Jaccard similarity when determining the similarities and differences between gene sequence data. The framework's highly parallel processing pipeline simultaneously identifies unique gene sequence words, minhashes each word to generate minhash signatures, and intersects minhash signatures to estimate Jaccard similarity for highly accurate and efficient identification of gene sequence taxonomy feature classes. These techniques are successfully used in Chapter 4 for rapid gene sequence classification and abundance estimation processes. Chapter 4 demonstrates application of the Collaborative Analytics Framework to develop the gene sequence classification software STRAND - "the Super-Threaded, Reference-Free, Alignment-Free, N-Sequence Decoder". STRAND is a machine learning platform for the identification and classification of gene sequence data into any number of gene sequence taxonomy classes using these highly parallel bioinformatics feature extraction techniques. I compare the accuracy and performance characteristics of Strand against RDP using 16S rRNA sequence data from the RDP training dataset and the Greengenes sequence repository. Strand produces comparable accuracy during ten-fold cross-validation performing classifications almost 20 times faster than RDP. Finally, a version of Strand is applied to the challenge of gene sequence abundance estimation. Very large volumes of gene sequence data from the National Center for Biotechnology Information are used as training examples for the classification of very short gene sequence reads or "snips" commonly produced by modern gene sequencing platforms. Strand is a highly flexible and scalable gene sequence classification program which can be implemented on multiple processors and or machines to accommodate very large gene sequence training and classification tasks.

In Chapter 5, I develop useful machine learning features for cybercrime. Website HTML, displayed text and screen shots are used as data input. I present methods to identify Ponzi and Escrow Fraud schemes as well as websites selling counterfeit goods. URL-level features, webpage-level features, and website-level features are combined into various machine learning models for the successful classification and clustering algorithms. I demonstrate the creation of large scale distance matrices and combine multiple distance matrices to identify and cluster together loose copies of replicated criminal websites as well.

Chapter 6 demonstrates an unsupervised machine learning approach for the identification of replicated criminal Ponzi Scheme and Escrow Fraud websites. In this research, I present a new technique called optimized combined clustering which links together replicated scam websites, even when the criminal has taken steps to hide connections. I explore topics such as feature extraction automation, including the extraction of rendered text, HTML structure, file structure and screen shots. I evaluated the method's applicability to cybercrime by measuring its performance against two collected datasets of scam websites: fake-escrow services and high-yield investment programs (HYIPs). This method is more accurate than general purpose consensus clustering approaches, as well as approaches designed for large-scale scams such as phishing that use more extensive copying of content.

Chapter 7 presents a highly parallel implementation for the identification of criminal websites. I identify and analyze websites selling counterfeit goods or knockoff products. URL-level, page-level, and website-level features are used, as well as webpage screenshots for identifying these criminal websites. A binary classifier predicts whether a given website is selling counterfeits by examining the automatically extracted features. We find that between January and August 2014, 32% of search results point to websites selling fakes overall.

9.2. Future Research Opportunities

In the future, I plan to apply both Strand and the Collaborative Analytics Framework to problems outside the domain of Bioinformatics. Since Strand uses no special encoding of text data, other unstructured input data sources should be evaluated. For example, high level testing on 9 classes of common malware data showed promising results with Strand predicting malware binaries at over 90% accuracy. This work should be extended, and Strand should be formally tested now that optimal hashing function sizes can be determined after reviewing both n-gram lengths and hash function bit sizes.

Two additional areas of Strand related research also appear highly favorable. First, the very simplistic parallelization pipelines developed for Strand appear highly compatible for optimization on a graphics card. The Chinese Search Engine company Baidu recently broke the world record for image recognition in early 2015. What was impressive about this accomplishment was that the "supercomputer" used for the task included only 32 GPUs [158]. In fact, Baidu cites a study which claims "that 12 GPUs in a 3-machine cluster can rival the performance of the performance of the 1,000-node CPU cluster behind the famous Google Brain project". [68]. GPU's may offer optimal performance for sequence classification unobtainable on a CPU. In addition, the multiple hashing operations performed during minhashing operations are favorable for GPU processing. Finally, embedded systems may offer great opportunity for the monetization of the Strand patent. Many opportunities for creating embedded system classification "black boxes" come to mind.

REFERENCES

[1] Alexa top 1 million websites. http://s3.amazonaws.com/alexa-static/top-1m.csv.zip.

[2] Spark. https://spark.apache.org/; Accessed: 2015-02-25.

[3] Abdi, H. *Encyclopedia of Measurement and Statistics*. SAGE Publications, Inc., 2007, pp. 598–605.

[4] Abdi, H., O'Toole, A., Valentin, D., and Edelman, B. Distatis: The analysis of multiple distance matrices. In *Computer Vision and Pattern Recognition - Workshops, 2005. CVPR Workshops. IEEE Computer Society Conference on* (June 2005), pp. 42–42.

[5] Abdi, H., Williams, L. J., Valentin, D., and Bennani-Dosse, M. Statis and distatis: optimum multitable principal component analysis and three way metric multidimensional scaling. *Wiley Interdisciplinary Reviews: Computational Statistics 4*, 2 (2012), 124–167.

[6] Abramson, K. D., Butts Jr, H. B., and Orbits, D. A. Affinity scheduling of processes on symmetric multiprocessing systems, Apr. 9 1996. US Patent 5,506,987.

[7] Abu-Nimeh, S., Nappa, D., Wang, X., and Nair, S. A comparison of machine learning techniques for phishing detection. In *Proceedings of the 2nd APWG eCrime Researchers Summit* (2007), ACM, pp. 60–69.

[8] Adams, M. D., Kelley, J. M., Gocayne, J. D., Dubnick, M., Polymeropoulos, M. H., Xiao, H., Merril, C. R., Wu, A., Olde, B., Moreno, R. F., et al. Complementary dna sequencing: expressed sequence tags and human genome project. *Science 252*, 5013 (1991), 1651–1656.

[9] Akerlof, G. A. The market for "lemons": Quality uncertainty and the market mechanism. *The Quarterly Journal of Economics 84*, 3 (1970), pp. 488–500.

[10] Alpha, E. Discovering the webâĂŹs hidden alpha, 2014. http://www.eaglealpha.com/whitepaper_pdf;Accessed:2014-12-08.

[11] Altschul, S. F., Gish, W., Miller, W., Myers, E. W., and Lipman, D. J. Basic local alignment search tool. *Journal of Molecular Biology 215*, 3 (1990), 403–410.

[12] Altschul, S. F., Gish, W., Miller, W., Myers, E. W., and Lipman, D. J. Basic local alignment search tool. *Journal of Molecular Biology 215*, 3 (1990), 403–410.

[13] Anderson, D. S., Fleizach, C., Savage, S., and Voelker, G. M. Spamscatter: Characterizing Internet scam hosting infrastructure. In *Proceedings of 16th USENIX Security Symposium* (Berkeley, CA, USA, 2007), USENIX Association, pp. 10:1–10:14.

[14] Anderson, R. Why information security is hard - an economic perspective. In *Proceedings of the 17th Annual Computer Security Applications Conference (ACSAC'01)* (New Orleans, LA, Dec. 2001).

[15] Anderson, R., and Moore, T. The economics of information security. *Science 314*, 5799 (Oct. 2006), 610–613.

[16] Apache. Hadoop, 2014. http://hadoop.apache.org/;AccessedOn:05/12/2015.

[17] Applications, A. C# tutorials, 2015. http://savvash.blogspot.com/p/c-tutorials.html;Accessed:2015-03-06.

[18] Bannur, S. N., Saul, L. K., and Savage, S. Judging a site by its content: learning the textual, structural, and visual features of malicious web pages. In *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence* (2011), ACM, pp. 1–10.

[19] Berenguel, L. P., and Reykjalin, J. Nonvolatile ramdisk memory, Aug. 31 1993. US Patent 5,241,508.

[20] Bik, E. M., Eckburg, P. B., Gill, S. R., Nelson, K. E., Purdom, E. A., Francois, F., Perez-Perez, G., Blaser, M. J., and Relman, D. A. Molecular analysis of the bacterial microbiota in the human stomach. *Proceedings of the National Academy of Sciences of the United States of America 103*, 3 (2006), 732–737.

[21] Blum, A., Wardman, B., Solorio, T., and Warner, G. Lexical feature based phishing url detection using online learning. In *Proceedings of the 3rd ACM Workshop on Artificial Intelligence and Security* (New York, NY, USA, 2010), AISec '10, ACM, pp. 54–60.

[22] Brenchley, R., Spannagl, M., Pfeifer, M., Barker, G. L., DâĂŹAmore, R., Allen, A. M., McKenzie, N., Kramer, M., Kerhornou, A., Bolser, D., et al. Analysis of the bread wheat genome using whole-genome shotgun sequencing. *Nature 491*, 7426 (2012), 705–710.

[23] Broder, A. Z. On the resemblance and containment of documents. In *Compression and Complexity of Sequences 1997. Proceedings* (1997), IEEE, pp. 21–29.

[24] Broder, A. Z., Charikar, M., Frieze, A. M., and Mitzenmacher, M. Min-wise independent permutations. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing* (1998), ACM, pp. 327–336.

[25] Broder, A. Z., Glassman, S. C., Manasse, M. S., and Zweig, G. Syntactic clustering of the web. *Computer Networks and ISDN Systems 29*, 8 (1997), 1157–1166.

[26] Buhler, J. Efficient large-scale sequence comparison by locality-sensitive hashing. *Bioinformatics 17*, 5 (2001), 419–428.

[27] Canali, D., Cova, M., Vigna, G., and Kruegel, C. Prophiler: a fast filter for the large-scale detection of malicious web pages. In *Proceedings of the 20th international conference on World wide web* (2011), ACM, pp. 197–206.

[28] Chandra, T. Sibyl: A system for large scale machine learning at google, 2014. https://www.youtube.com/watch?v=QoUVwGZb9tA;Accessed:2014-12-08.

[29] Chiu, C.-Y., Wang, H.-M., and Chen, C.-S. Fast min-hashing indexing and robust spatio-temporal matching for detecting video copies. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP) 6*, 2 (2010), 10.

[30] Chu, C., Kim, S. K., Lin, Y.-A., Yu, Y., Bradski, G., Ng, A. Y., and Olukotun, K. Map-reduce for machine learning on multicore. *Advances in neural information processing systems 19* (2007), 281.

[31] Chu, C. T., Kim, S. K., Lin, Y. A., Yu, Y., Bradski, G. R., Ng, A. Y., and Olukotun, K. Map-Reduce for Machine Learning on Multicore. In *NIPS* (2006), B. Schölkopf, J. C. Platt, and T. Hoffman, Eds., MIT Press, pp. 281–288.

[32] Chum, O., Perdoch, M., and Matas, J. Geometric min-hashing: Finding a (thick) needle in a haystack. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on* (2009), IEEE, pp. 17–24.

[33] Chum, O., Philbin, J., and Zisserman, A. Near duplicate image detection: min-hash and tf-idf weighting. In *BMVC* (2008), vol. 810, pp. 812–815.

[34] Clayton, R. WHOIS data extracted from templates (deft-whois), 2014. http://www.deft-whois.com.

[35] Clayton, R., and Mansfield, T. A study of whois privacy and proxy service abuse. In *13th Workshop on the Economics of Information Security* (2014).

[36] Culp, M., Johnson, K., and Michailidis, G. ada: An r package for stochastic boosting. *Journal of Statistical Software 17*, 2 (2006), 9.

[37] dartmouth. Pros and cons of openmp/mpi, 2011.

[38] de Souza, C. Near-duplicate image detection, 2014. http://www.codeproject.com/Articles/441226/Haar-feature-Object-Detection-in-Csharp;Accessed:2015-03-06.

[39] Dean, J., and Ghemawat, S. Mapreduce: simplified data processing on large clusters. *Communications of the ACM 51*, 1 (2008), 107–113.

[40] Dean, J., and Ghemawat, S. Mapreduce: A flexible data processing tool. *Communications of the ACM 53*, 1 (2010), 72–77.

[41] Der, M. F., Saul, L. K., Savage, S., and Voelker, G. M. Knock it off: Profiling the online storefronts of counterfeit merchandise. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (2014), ACM.

[42] Der, M. F., Saul, L. K., Savage, S., and Voelker, G. M. Knock it off: Profiling the online storefronts of counterfeit merchandise. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, NY, USA, 2014), KDD '14, ACM, pp. 1759–1768.

[43] Dimitriadou, E., Weingessel, A., and Hornik, K. A combination scheme for fuzzy clustering. *International Journal of Pattern Recognition and Artificial Intelligence 16*, 07 (2002), 901–912.

[44] Dingledine, R., Mathewson, N., and Syverson, P. Tor: The second-generation onion router. In *13th USENIX Security Symposium* (Aug. 2004).

[45] Documentation, R. Hierarchical clustering, 2015. https://stat.ethz.ch/R-manual/R-patched/library/stats/html/hclust.html;Accessed:2015-03-06.

[46] DOMO. Data never sleeps 2.0, 2013. http://www.domo.com/learn/data-never-sleeps-2;Accessed:2014-12-08.

[47] dotnetperls. C# interlocked, 2015. http://www.dotnetperls.com/interlocked;Accessed:2015-02-28.

[48] Drew, J. Mapreduce: Map reduction strategies using C#, 2013. http://www.codeproject.com/Articles/524233/MapReduceplus-fplusMapplusReductionplusStrategies;Accessed:2014-04-26.

[49] Drew, J. Wordreducer - example map reduction process that counts unique words in a body of text., 2013. http://www.jakemdrew.com/blog/mapreduce.htm;Accessed:2015-02-28.

[50] Drew, J. Clustering similar images using mapreduce style feature extraction with c# and r, 2014. http://blog.jakemdrew.com/2014/06/26/clustering-similar-images-using-mapreduce-style-feature-extraction-with-c-and ;Accessed:2015-03-06.

[51] Drew, J. Machine learning in parallel with support vector machines, generalized linear models, and adaptive boosting, 2014.

[52] Drew, J., and Hahsler, M. Strand: fast sequence comparison using mapreduce and locality sensitive hashing. In *Proceedings of the 5th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics* (2014), ACM, pp. 506–513.

[53] Duffy, J., and Essey, E. Running queries on multi-core processors, 2007. https://msdn.microsoft.com/en-us/magazine/cc163329.aspx;Accessed:2015-02-26.

[54] Eckburg, P. B., Bik, E. M., Bernstein, C. N., Purdom, E., Dethlefsen, L., Sargent, M., Gill, S. R., Nelson, K. E., and Relman, D. A. Diversity of the human intestinal microbial flora. *science 308*, 5728 (2005), 1635–1638.

[55] Edgar, R. C. Search and clustering orders of magnitude faster than BLAST. *Bioinformatics (Oxford, England) 26*, 19 (2010), 2460–1.

[56] Edgar, R. C. Search and clustering orders of magnitude faster than blast. *Bioinformatics 26*, 19 (2010), 2460–2461.

[57] Fiacco, A. V., and McCormick, G. P. *Nonlinear programming: sequential unconstrained minimization techniques*. No. 4. Siam, 1990.

[58] Florencio, D., and Herley, C. Evaluating a trial deployment of password re-use for phishing prevention. In *Second APWG eCrime Researchers Summit* (New York, NY, USA, 2007), eCrime '07, ACM, pp. 26–36.

[59] Gao, Z., Tseng, C.-h., Pei, Z., and Blaser, M. J. Molecular analysis of human forearm superficial skin bacterial biota. *Proceedings of the National Academy of Sciences 104*, 8 (2007), 2927–2932.

[60] Ghoting, A., Kambadur, P., Pednault, E., and Kannan, R. Nimble: a toolkit for the implementation of parallel data mining and machine learning algorithms on mapreduce. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining* (2011), ACM, pp. 334–342.

[61] Ghoting, A., Krishnamurthy, R., Pednault, E., Reinwald, B., Sindhwani, V., Tatikonda, S., Tian, Y., and Vaithyanathan, S. Systemml: Declarative machine learning on mapreduce. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on* (2011), IEEE, pp. 231–242.

[62] Gillick, D., Faria, A., and DeNero, J. Mapreduce: Distributed computing for machine learning. *Berkley, Dec 18* (2006).

[63] Gionis, A., Indyk, P., Motwani, R., et al. Similarity search in high dimensions via hashing. In *VLDB* (1999), vol. 99, pp. 518–529.

[64] Gordon, A., and Vichi, M. Fuzzy partition models for fitting a set of partitions. *Psychometrika 66*, 2 (2001), 229–247.

[65] Group, C. C. Part of speech tagging demo, 2014. http://cogcomp.cs.illinois.edu/page/demo_view/pos;AccessedOn:04/05/2015.

[66] hadooptutorial.info. Combiner in mapreduce, 2014. http://hadooptutorial.info/combiner-in-mapreduce/;AccessedOn:04/02/2015.

[67] Haile, A. Complete .net opencl implementations, 2013. http://stackoverflow.com/questions/5654048/complete-net-opencl-implementations;Accessed:2015-02-26.

[68] Harris, D. Baidu built a supercomputer for deep learning, 2015. https://gigaom.com/2015/01/14/baidu-has-built-a-supercomputer-for-deep-learning/;AccessedOn:10/18/2015.

[69] Hornik, K. A clue for cluster ensembles.

[70] Hothorn, T. Cran task view: Machine learning and statistical learning, 2014. http://cran.r-project.org/web/views/MachineLearning.html;Accessed:2014-12-08.

[71] Hyman, R. W., Fukushima, M., Diamond, L., Kumm, J., Giudice, L. C., and Davis, R. W. Microbes on the human vaginal epithelium. *Proceedings of the National Academy of Sciences 102*, 22 (2005), 7952–7957.

[72] Icaza, M. d. e. a. Mono c# compiler, 2015. http://www.mono-project.com/docs/about-mono/languages/csharp/;Accessed:2015-11-02.

[73] (IDC), I. D. C. The digital universe in 2020, 2013. http://www.emc.com/collateral/analyst-reports/idc-digital-universe-united-states.pdf;Accessed:2014-12-08.

[74] Indyk, P., and Motwani, R. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing* (1998), ACM, pp. 604–613.

[75] Intel. Why use intelÂő cilkâĎć plus?, 2015. https://www.cilkplus.org/;Accessed:2015-02-26.

[76] Ioffe, S. Improved consistent sampling, weighted minhash and l1 sketching. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on* (2010), IEEE, pp. 246–255.

[77] John, J. P., Yu, F., Xie, Y., Krishnamurthy, A., and Abadi, M. deseo: Combating search-result poisoning. In *USENIX Security Symposium* (2011), USENIX Association.

[78] Johnson, S. C. Hierarchical clustering schemes. *Psychometrika 32*, 3 (1967), 241–254.

[79] Kaggle. Microsoft malware classification challenge (big 2015), 2015. https://www.kaggle.com/c/malware-classification;Accessed:2015-11-04.

[80] Kaggle. Microsoft malware winners' interview: 1st place, "no to overfitting", 2015. http://blog.kaggle.com/2015/05/26/microsoft-malware-winners-interview-1st-place-no-to-overfitting/;Accessed:2015-11-02.

[81] Kammerstetter, M., Platzer, C., and Wondracek, G. Vanity, cracks and malware: Insights into the anti-copy protection ecosystem. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security* (New York, NY, USA, 2012), CCS '12, ACM, pp. 809–820.

[82] Kanich, C., Kreibich, C., Levchenko, K., Enright, B., Voelker, G., Paxson, V., and Savage, S. Spamalytics: An empirical analysis of spam marketing conversion. In *Conference on Computer and Communications Security (CCS)* (Alexandria, VA, Oct. 2008).

[83] Kaplan, E., and Meier, P. Nonparametric estimation from incomplete observations. *Journal of the American Statistical Association 53* (1958), 457–481.

[84] Keco, D., and Subasi, A. Parallelization of genetic algorithms using hadoop map/reduce. *SouthEast Europe Journal of Soft Computing 1*, 2 (2012).

[85] Kent, W. J. Blat-the blast-like alignment tool. *Genome research 12*, 4 (2002), 656–664.

[86] Kmett, E. Near-duplicate image detection, 2009. http://stackoverflow.com/questions/1034900/near-duplicate-image-detection/1076507#1076507;Accessed:2015-03-06.

[87] Krolak-Schwerdt, S. Three-way multidimensional scaling: Formal properties and relationships between scaling methods. In *Data Analysis and Decision Support*, D. Baier, R. Decker, and L. Schmidt-Thieme, Eds., Studies in Classification, Data Analysis, and Knowledge Organization. Springer Berlin Heidelberg, 2005, pp. 82–90.

[88] Langfelder, P., Zhang, B., and Horvath, S. Defining clusters from a hierarchical cluster tree. *Bioinformatics 24*, 5 (Mar. 2008), 719–720.

[89] Layton, R., Watters, P., and Dazeley, R. Automatically determining phishing campaigns using the uscap methodology. In *eCrime Researchers Summit (eCrime), 2010* (Oct 2010), pp. 1–8.

[90] Leontiadis, N., Moore, T., and Christin, N. Measuring and analyzing search-redirection attacks in the illicit online prescription drug trade. In *USENIX Security Symposium* (2011).

[91] Leontiadis, N., Moore, T., and Christin, N. Pick your poison: pricing and inventories at unlicensed online pharmacies. In *Proceedings of the fourteenth ACM conference on Electronic commerce* (2013), ACM, pp. 621–638.

[92] Leontiadis, N., Moore, T., and Christin, N. A nearly four-year longitudinal study of search-engine poisoning. In *Proceedings of ACM CCS 2014* (Scottsdale, AZ, Nov. 2014).

[93] Leskovec, J., Rajaraman, A., and Ullman, J. D. *Mining of massive datasets*. Cambridge University Press, 2014.

[94] Levchenko, K., Chachra, N., Enright, B., Felegyhazi, M., Grier, C., Halvorson, T., Kanich, C., Kreibich, C., Liu, H., McCoy, D., Pitsillidis, A., Weaver, N., Paxson, V., Voelker, G., and Savage, S. Click trajectories: End-to-end analysis of the spam value chain. In *Proceedings of IEEE Security and Privacy* (Oakland, CA, May 2011).

[95] Levchenko, K., Pitsillidis, A., Chachra, N., Enright, B., Félegyházi, M., Grier, C., Halvorson, T., Kanich, C., Kreibich, C., Liu, H., McCoy, D., Weaver, N., Paxson, V., Voelker, G. M., and Savage, S. Click trajectories: End-to-end analysis of the spam value chain. In *Proceedings of the 2011 IEEE Symposium on Security and Privacy* (Washington, DC, USA, 2011), SP '11, IEEE Computer Society, pp. 431–446.

[96] Ley, R. E., Bäckhed, F., Turnbaugh, P., Lozupone, C. A., Knight, R. D., and Gordon, J. I. Obesity alters gut microbial ecology. *Proceedings of the National Academy of Sciences of the United States of America 102*, 31 (2005), 11070–11075.

[97] Ley, R. E., Turnbaugh, P. J., Klein, S., and Gordon, J. I. Microbial ecology: human gut microbes associated with obesity. *Nature 444*, 7122 (2006), 1022–1023.

[98] Li, L., Wang, D., Li, T., Knox, D., and Padmanabhan, B. Scene: A scalable two-stage personalized news recommendation system. In *ACM Conference on Information Retrieval (SIGIR)* (2011).

[99] Lin, J.-L. Detection of cloaked web spam by using tag-based methods. *Expert Syst. Appl. 36*, 4 (May 2009), 7493–7499. Available at http://dx.doi.org/10.1016/j.eswa.2008.09.056.

[100] Littlejohn, C., Baldacchino, A., Schifano, F., and Deluca, P. Internet pharmacies and online prescription drug sales: a cross-sectional study. *Drugs: Education, Prevention, and Policy 12*, 1 (2005), 75–80.

[101] Low, Y., Gonzalez, J. E., Kyrola, A., Bickson, D., Guestrin, C. E., and Hellerstein, J. Graphlab: A new framework for parallel machine learning. *arXiv preprint arXiv:1408.2041* (2014).

[102] Ma, J., Saul, L. K., Savage, S., and Voelker, G. M. Beyond blacklists: learning to detect malicious web sites from suspicious urls. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining* (2009), ACM, pp. 1245–1254.

[103] Marcais, G., and Kingsford, C. A fast, lock-free approach for efficient parallel counting of occurrences of k-mers. *Bioinformatics 27*, 6 (2011), 764–770.

[104] Mavrommatis, N. P. P., and Monrose, M. A. R. F. All your iframes point to us. In *USENIX Security Symposium* (2008), pp. 1–16.

[105] McCoy, D., Pitsillidis, A., Jordan, G., Weaver, N., Kreibich, C., Krebs, B., Voelker, G., Savage, S., and Levchenko, K. Pharmaleaks: Understanding the business of online pharmaceutical affiliate programs. In *Proceedings of USENIX Security 2012* (Bellevue, WA, Aug. 2012).

[106] McKenna, A., Hanna, M., Banks, E., Sivachenko, A., Cibulskis, K., Kernytsky, A., Garimella, K., Altshuler, D., Gabriel, S., Daly, M., et al. The genome analysis toolkit: A mapreduce framework for analyzing next-generation dna sequencing data. *Genome research 20*, 9 (2010), 1297–1303.

[107] Meyer, D. Support vector machines. *The Interface to libsvm in package e1071. e1071 Vignette* (2012).

[108] Microsoft. Pipelines, 2013. https://msdn.microsoft.com/en-us/library/ff963548.aspx;AccessedOn:03/31/2015.

[109] Microsoft. Numa support, 2015. https://msdn.microsoft.com/en-us/library/windows/desktop/aa363804(v=vs.85).aspx;Accessed:2015-02-26.

[110] Microsoft. Readerwriterlock class, 2015. http://msdn.microsoft.com/en-us/library/system.threading.readerwriterlock.aspx;Accessed:2015-02-28.

[111] Migliore, M. Similar image finder, 2015. https://similarimagesfinder.codeplex.com/;Accessed:2015-03-06.

[112] Moore, T., and Clayton, R. Examining the impact of website take-down on phishing. In *Second APWG eCrime Researchers Summit* (Pittsburgh, PA, Oct. 2007), eCrime '07, ACM.

[113] Moore, T., and Clayton, R. *The Impact of Incentives on Notice and Take-down*. Springer, 2008, pp. 199–223.

[114] Moore, T., Han, J., and Clayton, R. The postmodern Ponzi scheme: Empirical analysis of high-yield investment programs. In *Financial Cryptography* (2012), A. D. Keromytis, Ed., vol. 7397 of *Lecture Notes in Computer Science*, Springer, pp. 41–56.

[115] Moore, T., Leontiadis, N., and Christin, N. Fashion crimes: trending-term exploitation on the web. In *ACM Conference on Computer and Communications Security* (2011), Y. Chen, G. Danezis, and V. Shmatikov, Eds., ACM, pp. 455–466.

[116] Needleman, S. B., and Wunsch, C. D. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology 48*, 3 (1970), 443–453.

[117] Neisius, J., and Clayton, R. Orchestrated crime: The high yield investment fraud ecosystem. In *APWG Symposium on Electronic Crime Research* (2014).

[118] Ntoulas, A., Najork, M., Manasse, M., and Fetterly, D. Detecting spam web pages through content analysis. In *Proceedings of the 15th international conference on World Wide Web* (2006), ACM, pp. 83–92.

[119] openmp.org. What problem does openmp solve ?, 2008. http://openmp.org/openmp-faq.html#Problems;Accessed:2015-02-26.

[120] Oshana, R. Multicore software development, smu âĂŞ fall 2013, 2008.

[121] Ounit, R., Wanamaker, S., Close, T. J., and Lonardi, S. Clark: fast and accurate classification of metagenomic and genomic sequences using discriminative k-mers. *BMC genomics 16*, 1 (2015), 236.

[122] Ounit, R., Wanamaker, S., Close, T. J., and Lonardi, S. Clark: fast and accurate classification of metagenomic and genomic sequences using discriminative k-mers supplementary material. *BMC genomics 16*, 1 (2015), 236.

[123] Overflow, S. Image fingerprint, 2009. http://stackoverflow.com/questions/596262/image-fingerprint-to-compare-similarity-of-many-images;Accessed:2014-06-24.

[124] Pei, Z., Bini, E. J., Yang, L., Zhou, M., Francois, F., and Blaser, M. J. Bacterial biota in the human distal esophagus. *Proceedings of the National Academy of Sciences 101*, 12 (2004), 4250–4255.

[125] Perls, D. N. C# string memory, 2014. http://www.dotnetperls.com/string-memory;AccessedOn:10/18/2015.

[126] Peter Wang, I. Compare windows* threads, openmp*, intelÂő threading building blocks for parallel programming, 2008. http://software.intel.com/en-us/blogs/2008/12/16/compare-windows-threads-openmp-intel-threading-building-blocks-for-parallel-p;Accessed:2015-02-26.

[127] Provos, N., Mavrommatis, P., Rajab, M., and Monrose, F. All your iFrames point to us. In *17th USENIX Security Symposium* (Aug. 2008).

[128] Pruitt, K. D., Tatusova, T., Brown, G. R., and Maglott, D. R. Ncbi reference sequences (refseq): current status, new features and genome annotation policy. *Nucleic acids research 40*, D1 (2012), D130–D135.

[129] Rajaraman, A., and Ullman, J. *Mining of Massive Datasets*. Mining of Massive Datasets. Cambridge University Press, 2012.

[130] Rand, W. M. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association 66*, 336 (1971), 846–850.

[131] Reid, A. D., Ford, S. A., and Lin, Y. Analyzing and transforming a computer program for executing on asymmetric multiprocessing systems, Sept. 11 2007. US Patent App. 11/898,360.

[132] Riek, M., Boehme, R., and Moore, T. Understanding the influence of cybercrime risk on the e-service adoption of European Internet users. In *13th Workshop on the Economics of Information Security* (2014).

[133] Rodriguez, G. Generalized linear models, 2014. http://data.princeton.edu/R/glms.html;Accessed:2014-04-19.

[134] Roth, D., and Zelenko, D. Part of speech tagging using a network of linear separators. In *Coling-Acl, The 17th International Conference on Computational Linguistics* (1998), pp. 1136–1142.

[135] Russinovich, M. Pushing the limits of windows: Processes and threads, 2009. http://blogs.technet.com/b/markrussinovich/archive/2009/07/08/3261309.aspx;Accessed:2015-02-28.

[136] Shannon, C. E. A mathematical theory of communication. *The Bell Systems Technical Journal 27* (1948), 379–423.

[137] Smith, M. D., and Telang, R. Competing with free: The impact of movie broadcasts on dvd sales and internet piracy1. *MIS Q. 33*, 2 (June 2009), 321–338.

[138] Smith, T. F., and Waterman, M. S. Identification of common molecular subsequences. *Journal of Molecular Biology 147*, 1 (March 1981), 195–197.

[139] SRIDHAR PAPPU, H. P. To handle the big data deluge, hp plots a giant leap forward, 2014. https://ssl.www8.hp.com/hpmatter/issue-no-1-june-2014/handle-big-data-deluge-hp-plots-giant-leap-forward?jumpid=sc_pur3bc5n8v/dm:_N5823.186294OUTBRAININC_109138141_282642904_0_2879120;Accessed:2014-12-08.

[140] Toub, S. Patterns of parallel programming-understanding and applying parallel patterns with the .net framework 4 and visual c#. *Parallel Computing Platform, Microsoft Corporation. Version (February 16, 2010)* (2010).

[141] Turnbaugh, P. J., Ley, R. E., Hamady, M., Fraser-Liggett, C. M., Knight, R., and Gordon, J. I. The human microbiome project. *Nature 449*, 7164 (2007), 804–810.

[142] Tyson, G. W., Chapman, J., Hugenholtz, P., Allen, E. E., Ram, R. J., Richardson, P. M., Solovyev, V. V., Rubin, E. M., Rokhsar, D. S., and Banfield, J. F. Community structure and metabolism through reconstruction of microbial genomes from the environment. *Nature 428*, 6978 (2004), 37–43.

[143] Ukkonen, E. Approximate string-matching with q-grams and maximal matches. *Theoretical Computer Science 92*, 205 (1992), 191–211.

[144] Urvoy, T., Chauveau, E., Filoche, P., and Lavergne, T. Tracking web spam with html style similarities. *ACM Trans. Web 2*, 1 (Mar. 2008), 3:1–3:28.

[145] Venter, J. C., Remington, K., Heidelberg, J. F., Halpern, A. L., Rusch, D., Eisen, J. A., Wu, D., Paulsen, I., Nelson, K. E., Nelson, W., et al. Environmental genome shotgun sequencing of the sargasso sea. *science 304*, 5667 (2004), 66–74.

[146] Vinga, S., and Almeida, J. Alignment-free sequence comparison — A review. *Bioinformatics 19*, 4 (2003), 513–523.

[147] Wang, D., Savage, S., and Voelker, G. Cloak and dagger: Dynamics of web search cloaking. In *Proceedings of the 18th ACM Conference on Computer and Communications Security* (2011), ACM, pp. 477–490.

[148] Wang, D. Y., Der, M., Karami, M., Saul, L., McCoy, D., Savage, S., and Voelker, G. M. Search + seizure: The effectiveness of interventions on SEO campaigns. In *ACM Internet Measurement Conference (IMC)* (2014), ACM.

[149] Wang, D. Y., Savage, S., and Voelker, G. M. Cloak and dagger: dynamics of web search cloaking. In *Proceedings of the 18th ACM conference on Computer and communications security* (New York, NY, USA, 2011), CCS '11, ACM, pp. 477–490. Available at http://doi.acm.org/10.1145/2046707.2046763.

[150] Wang, L. Microsoft malware classification challenge (big 2015) first place team: Say no to overfitting, 2015. https://github.com/xiaozhouwang/kaggle_Microsoft_Malware/blob/master/Saynotooverfitting.pdf;Accessed:2015-11-02.

[151] Wang, Q., Garrity, G. M., Tiedje, J. M., and Cole, J. R. Naive bayesian classifier for rapid assignment of RNA sequences into the new bacterial taxonomy. *Applied and Environmental Microbiology 73*, 16 (2007), 5261–5267.

[152] Wardman, B., and Warner, G. Automating phishing website identification through deep md5 matching. In *eCrime Researchers Summit, 2008* (2008), IEEE, pp. 1–7.

[153] Webb, S., Caverlee, J., and Pu, C. Predicting web spam with http session information. In *Proceedings of the 17th ACM conference on Information and knowledge management* (2008), ACM, pp. 339–348.

[154] Wernersson, R., Schierup, M. H., Jørgensen, F. G., Gorodkin, J., Panitz, F., Stærfeldt, H.-H., Christensen, O. F., Mailund, T., Hornshøj, H., Klein, A., et al. Pigs in sequence space: a 0.66 x coverage pig genome survey based on shotgun sequencing. *BMC genomics 6*, 1 (2005), 70.

[155] Wikipedia. Luminance, 2015. http://en.wikipedia.org/wiki/Luminance;Accessed:2014-06-24.

[156] Wikipedia. Relative luminance, 2015. http://en.wikipedia.org/wiki/Luminance_(relative)#cite_note-1;Accessed:2014-06-24.

[157] Wood, D. E., and Salzberg, S. L. Kraken: ultrafast metagenomic sequence classification using exact alignments. *Genome Biol 15*, 3 (2014), R46.

[158] Wu, R. Deep image, 2015. https://youtu.be/NLyYG_ih_ak;AccessedOn:10/18/2015.

[159] Xu, J. Opencl–the open standard for parallel programming of heterogeneous systems.

[160] Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., and Stoica, I. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing* (2010), pp. 10–10.