SPAM, PHISHING, AND FRAUD DETECTION USING RANDOM PROJECTIONS, ADVERSARIAL LEARNING, AND SEMI-SUPERVISED LEARNING

by

David DeBarr

A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
in Partial Fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Computer Science

Committee:

_____	Dr. Harry Wechsler, Dissertation Director
_____	Dr. Jessica Lin, Committee Member
_____	Dr. Duminda Wijesekera, Committee Member
_____	Dr. Mihai Boicu, Committee Member
_____	Dr. Sanjeev Setia, Department Chair
_____	Dr. Kenneth Ball, Dean, Volgenau School of Engineering
Date: _____	Fall Semester 2013 George Mason University Fairfax, VA

Spam, Phishing, and Fraud Detection Using Random Projections, Adversarial Learning, and Semi-Supervised Learning

A Dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

by

David DeBarr
Master of Science
George Mason University, 2006

Director: Harry Wechsler, Professor
Department of Computer Science

Fall Semester 2013
George Mason University
Fairfax, VA

DEDICATION

This work is dedicated to my family, particularly my wife Kathie and my daughters Tiffany and Melissa, who supported me during this research. I also dedicate this to my brother, mother, and sister, without whom this work would not have been possible.

ACKNOWLEDGEMENTS

I would like to thank my advisor Harry Wechsler for both his patience and guidance. He has provided invaluable advice for both computer science research and life. I am also grateful to the many researchers who have come before me, allowing me to learn from their work. Last, but not least, I am grateful to my fellow researchers, who continue to inspire me to think critically, ask questions, and seek answers (including Venkatesh Ramanathan and Hao Sun).

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF EQUATIONS

LIST OF ABBREVIATIONS OR SYMBOLS

Association for the Advancement of Artificial Intelligence ..... AAAI
Association for Computing Machinery ..... ACM
Also Known As ..... AKA
Anti-Phishing Working Group ..... APWG
Adaptively Resample and Combine ..... ARC
Advanced Research Projects Agency ..... ARPA
American Statistical Association ..... ASA
American Standard Code for Information Interchange ..... ASCII
Area Under the Curve ..... AUC
Bootstrap Aggregation ..... BAG
Classification And Regression Tree ..... CART
Conference on E-mail and Anti-Spam ..... CEAS
Conference on Intelligent Text Processing and Computational Linguistics ..... CICLing
Conference on Information and Knowledge Management ..... CIKM
Conference on Learning Theory ..... COLT
Electronic Mail ..... E-mail
Expectation Maximization ..... EM
Empirical Risk Minimization ..... ERM
False Negative ..... FN
False Positive ..... FP
Gradient Boosting Machine ..... GBM
International Conference on Machine Learning ..... ICML
International Conference on Machine Learning and Applications ..... ICMLA
Inverse Document Frequency ..... IDF
Institute for Electrical and Electronics Engineers ..... IEEE
Internet Engineering Task Force ..... IETF
If and only if ..... IFF
Incorporated ..... Inc
Information Retrieval ..... IR
Intelligence and Security Informatics ..... ISI
Journal of Machine Learning Research ..... JMLR
Knowledge Discovery and Data Mining ..... KDD
Karush, Kuhn, and Tucker ..... KKT
Latent Dirichlet Allocation ..... LDA
Markov Chain Monte Carlo ..... MCMC
Multipurpose Internet Mail Extensions ..... MIME

Massachusetts Institute of Technology ..... MIT
Mail Transfer Agent ..... MTA
Neural Information Processing Systems ..... NIPS
Out Of Bag ..... OOB
Partitioning Around Medoids ..... PAM
Pattern Analysis, Statistical Modeling, and Computational Learning ..... PASCAL
Random Forest ..... RF
Randomized Hough Transform ..... RHT
Receiver Operating Characteristic ..... ROC
Reject On Negative Impact ..... RONI
Social computing, Behavioral modeling, and Prediction ..... SBP
Special Interest Group on Information Retrieval ..... SIGIR
Special Interest Group on Knowledge Discovery and Data mining ..... SIGKDD
Subspace, Latent Structure, and Feature Selection ..... SLSFS
Semi-Supervised Learning ..... SSL
Synthetic Minority Over-sampling Technique ..... SMOTE
Simple Mail Transfer Protocol ..... SMTP
Social Network Analysis ..... SNA
Structural Risk Minimization ..... SRM
Semi-Supervised Learning ..... SSL
Singular Value Decomposition ..... SVD
Support Vector Machine ..... SVM
Term Frequency ..... TF
Topic Modeling Toolbox ..... TMT
True Negative ..... TN
True Positive ..... TP
Text Retrieval Conference ..... TREC
Uniform Resource Locator ..... URL
Vapnik-Chervonenkis VC

ABSTRACT

SPAM, PHISHING, AND FRAUD DETECTION USING RANDOM PROJECTIONS, ADVERSARIAL LEARNING, AND SEMI-SUPERVISED LEARNING

David DeBarr, Ph.D.

George Mason University, 2013

Dissertation Director: Dr. Harry Wechsler

Spam, phishing, and fraud detection are security applications that impact most people. Challenges for building spam, phishing, and fraud detection models include difficulty in obtaining annotated data, increased computational complexity for robust detection methods, annotation errors, and changes in the underlying data distribution. We address the above challenges as follows. Clustering and active learning are combined to make efficient use of annotated data, yielding state of the art spam detection performance using only 10% of the annotated data employed by previously published methods. Social Network Analysis (SNA) reputation features for mail transfer agents are introduced to evaluate paths from sender to receiver, increasing the detection rate by 70% (with the same false positive rate) for state of the art spam detection. Random projections with boosting achieve state of the art spam detection with a 75% reduction in computational cost for message classification. The Randomized Hough Transform - Support Vector

Machine updates training set annotations, increasing the (precision, recall) F measure by 9.3% compared to a state of the art method for handling adversarial noise. Spectral clustering of URL n-grams and transductive semi-supervised learning are used to increase the detection rate by 100% (doubling the detection rate with the same false positive rate) for state of the art phishing detection under adversarial modification of message text. Reputation and similarity features are used to enhance the ability to withstand changes in underlying data distributions, producing a 13.5% increase in cost savings for state of the art fraud detection.

1. INTRODUCTION

This research evaluates enhanced representations and methods for computer-based security applications. This chapter briefly introduces the security tasks, associated challenges, proposed solutions, performance of the proposed solutions, and major contributions for this research.

1.1 Security Tasks

The security tasks addressed in this research include spam detection, phishing detection, and fraud detection. The first two applications focus on detection based on electronic mail (e-mail) messages, while the last application focuses on detection based on insurance claim information. Spam detection involves identifying unsolicited "junk" mail. This includes unwanted advertisements, scam solicitations, and malware infection attempts transmitted via electronic mail (e-mail). Phishing detection involves identifying identity theft attempts. The message sender solicits account or other identity information either directly via reply message or via a web site contacted via a link found in the message. Fraud detection involves identifying the use of deceit for financial gain, as in fraudulent auto insurance claims.

These applications involve high volumes of streaming data. The sheer volume of transactions for these applications requires providers to use automated means to detect

undesirable transactions, because it would be cost prohibitive to hire humans to review all transactions. A public e-mail service, such as Microsoft's Hotmail service, may receive billions of electronic mail (e-mail) messages per day, while an auto insurance company may handle tens of thousands of insurance claims per year. The e-mail service providers would like to efficiently detect and block spam messages and phishing messages, while the auto insurance companies would like to efficiently detect and reject fraudulent claims.

1.2 Challenges

The major challenges involved in security applications include difficulty obtaining annotated data and responding to adversarial actions designed to avoid detection. Annotated data is required for training machine learning detectors and to periodically evaluate system performance. It often costs money to get humans to label examples for training and testing. Furthermore, humans may make errors when labeling the data. We need to be intelligent about which examples we select for labeling and check the labels for consistency with other labeled examples.

Adversaries are motivated to change their representation to avoid detection. So even if a perfect detector is deployed, an adversary has financial motivation for learning how to quickly subvert the deployed detector. This means our solutions must adapt to adversarial challenges. Content filters ignore e-mail server reputation when attempting to identify spam messages. Both content and mail server reputation information can be used to enhance detection.

Learning methods which are robust against minor changes in the underlying data distributions are computationally expensive to implement. Computationally inexpensive methods are needed to reduce costs while maintaining robustness.

In some circumstances, adversaries may be able to add noise to annotated labels. For example, a public e-mail service provider may solicit users to mark messages as "spam" or "not spam". It's necessary to ensure the labels provided are consistent with existing data.

Content filtering, based on methods such as Latent Dirichlet Allocation (LDA), relies on content to identify phishing messages. When not much content is present, the content filter will likely fail to properly detect phishing. A new approach is needed to evaluate similarity of an incoming message to previously observed messages.

Finally, changes in the underlying data distributions can degrade performance of a classifier over time. A new approach is needed to incorporate feedback from recent observations into the detector's decision function.

1.3 Solutions/Methods

Several solutions to these challenges are proposed and evaluated in this research. Clustering and active learning are evaluated for the ability to reduce the number of annotated observations required for state of the art spam detection. Social Network Analysis (SNA) features are evaluated for the ability to increase the accuracy of the spam detector. Random subspace projections are used with boosting to produce an efficient spam detector that is robust against small changes in the underlying data distributions.

Spectral clustering with transduction is used to effectively identify phishing messages with reduced content. Finally, reputation and similarity features are used to increase the cost savings for state of the art fraud detection.

1.4 Performance

Clustering and active learning achieves results that are comparable to state of the art results with a 90% reduction in the amount of annotated data required for training a spam detector [1]. Social Network Analysis (SNA) features resulted in a 70% increase in the detection rate for spam when combined with content filtering for spam detection [2]. Random subspace projection with boosting decreases the computational complexity by 75% for state of the art spam detection [3]. Repeated random sampling and voting to identify mislabeled training examples increases the F measure by 9.3% for spam detection under adversarial conditions [4]. Spectral clustering with transduction increases the detection rate by 100% for state of the art phishing detection under adversarial message modification [5]. Reputation and similarity features increase the cost savings by 13.5% for state of the art fraud detection [6]. Performance information for existing solutions can be found in the related references [7] [8] [9].

1.5 Contributions

Clustering and actively learning demonstrated the utility of combining and applying existing techniques for clustering (exploration) and uncertainty sampling (exploitation) to spam detection. This advances the state of the art by reducing the

amount of data required to construct a state of the art spam detection model. Social Network Analysis (SNA) features for mail transfer agents were introduced for spam detection. This advances the state of the art by improving the detection rate for spam detection, compared to content filtering alone. Random subspace projections with boosting were introduced for spam detection. This advances the state of the art by introducing a new learning method that significantly reduces computational complexity for state of the art spam detection. The difference between Random Boost and Logit Boost is analogous to the difference between a Random Forest and a bagged decision tree ensemble. The use of voting to identify mislabeled training examples was introduced for adversarial spam detection. The Randomized Hough Transform – Support Vector Machine advances the state of the art by introducing a new learning method that offers better (precision, recall) response to adversarial annotation noise than a state of the art method. Spectral clustering of Uniform Resource Locator (URL) n-grams and transductive semi-supervised learning were introduced for adversarial phishing detection. This advances the state of the art by introducing features that significantly improve detection response to adversarial message modification compared to a state of the art method. Reputation and similarity features were introduced for state of the art fraud detection. This enhanced representation advances the state of the art in fraud detection by significantly increasing the cost savings metric compared to a state of the art method. Table 1 summarizes these contributions.

Table 1 Summary of Contributions

Challenge	Solution	Performance/Impact	What's New?
Annotation is expensive	Clustering and active learning	90% reduction in annotated data	Representation
Content filters ignore server reputation	SNA reputation features	70% increase in detection rate	Representation
Random Forests are computationally expensive	Random Boost	75% reduction in computational cost	Method
Annotation may contain noise	RHT-SVM	9.3% increase in F measure (adversarial annotation noise)	Method
Content filters rely on contents for classification	Spectral clustering and transductive SSL	100% increase in detection rate (adversarial message modification)	Representation
Changes in data distribution	Reputation and similarity features	13.5% increase in cost savings	Representation

The remainder of this thesis is organized as follows. Chapter 2 briefly describes background material related to the selected security applications. Chapter 3 discusses the machine learning methods used in this research. Chapter 4 defines the metrics used to evaluate this research. Chapter 5 presents research related to spam detection. Chapter 6 presents research related to phishing detection. Chapter 7 presents research related to fraud detection. Chapter 8 discusses the results, and Chapter 9 concludes the thesis with a recap of the results.

2. BACKGROUND

The major security applications addressed in this work include spam detection, phishing detection, and fraud detection. This chapter discusses these applications and commonly used representations for data in these domains.

2.1 Spam Detection

Spam is unwanted, unsolicited e-mail. Symantec's "Message Labs Intelligence" report for August 2010 [10] estimated over 92% of e-mail messages to be spam. Spammers are encouraged to distribute their messages broadly as this increases the probability of achieving their goals. This includes advertisements for a variety of products and services, such as pharmaceuticals, jewelry, electronics, software, loans, stocks, gambling, weight loss, and pornography, as well as malware distribution attempts. Total revenue for spam-advertised goods has been estimated to be hundreds of millions of dollars per year [11], with relatively low costs for spammers.

2.2 Phishing Detection

Phishing is a form of identity theft. A typical phishing attempt consists of a phisher sending an e-mail to a user. The e-mail appears to come from a legitimate service, such as a financial institution, a social networking web site, or an electronic

message service provider. The e-mail contains a link to a web site that mimics the web site of the legitimate service provider. In fact, the graphics and layout of the web site may be identical to the web site of the legitimate service provider. The only difference may be the use of an intermediate link for grabbing form related data, such as account login information. This information is then passed to the legitimate service provider's web site making it difficult for the user to know their account information has just been compromised. The phisher then exploits the victim's compromised financial accounts, or the victim's social contacts. The victim's social contacts potentially become the next target of the phisher.

With access to the victim's account, and a list of the victim's social contacts, the phisher can now engage in spear phishing. Rather than sending phishing messages as unsolicited messages to randomly selected addresses harvested from Internet sources, the phisher can now direct messages to specific individuals. In this form of spear phishing, brief messages are sent to known contacts along with a link to the phishing web site. Since the message will come from a known contact, individuals may be much more likely to trust the link. This can be especially damaging if the initial victim is an administration account for a service provider. Because most phishing detection systems rely on content filtering techniques, such as Latent Dirichlet Allocation (LDA), many phishing detection systems will fail to identify the message as a phishing attempt.

2.3 Message Representation

The Simple Mail Transfer Protocol (SMTP) [12] is used to transmit e-mail messages. These messages conform to the format specifications of Request For Comments (RFC) 822 [13]. The inmail.38377 message from the TREC 2007 Spam Track Corpus is shown in the Figure 1 Example E-mail Message.

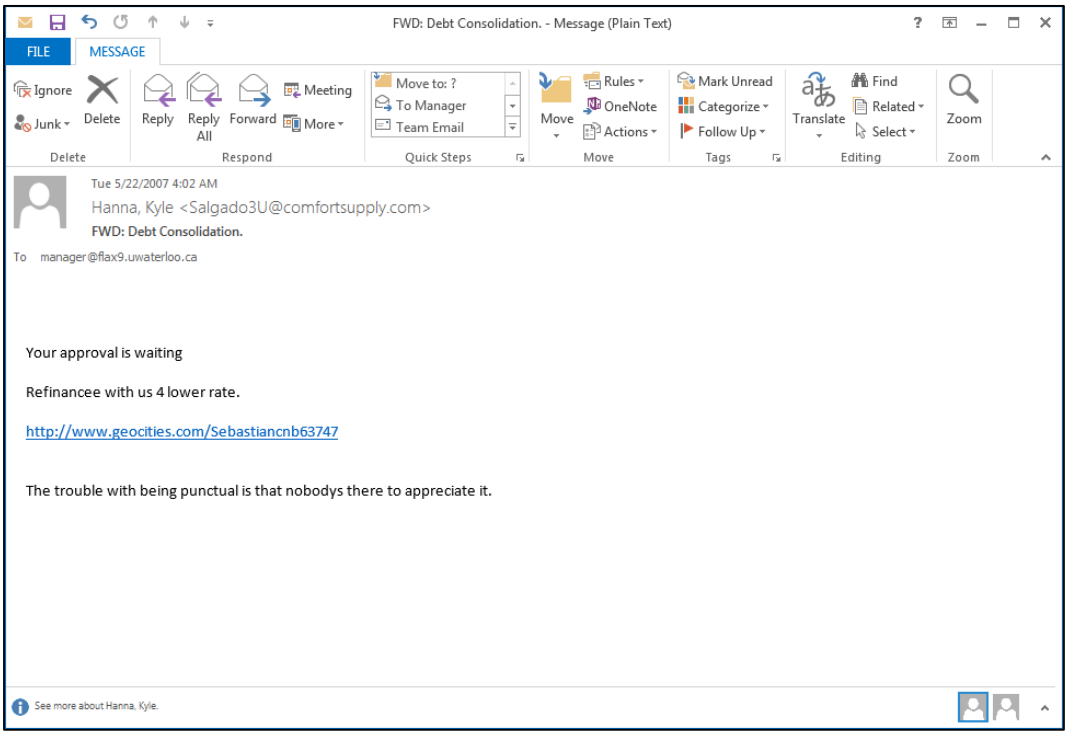

Figure 1 Example E-mail Message

The actual text of this message is shown in the Figure 2 Example E-mail Message Text. The first line records the e-mail address of the originator as reported to the mail server, along with the timestamp of receipt. This is followed by a series of "headers",

lines identified by a header name followed by a colon and the assigned value. Continuation lines in the header section start with a "white space" character; i.e. a space (ASCII 32) or a horizontal tab (ASCII 9). There are 10 headers in this example. The "return-path" header is an e-mail address to be used for "replies." The "received" header identifies the identity of the computer that transmitted this message to the destination server. There can be multiple "received" headers; e.g. the originator's computer may send a message to a list server that forwards the message to other servers. The "message-id" header is simply a reference identifier for this particular message. The "from" header is supposed to identify the message sender, but it may actually contain arbitrary text. The "to" header is supposed to identify the message recipient, but it may also contain arbitrary text. The actual recipient's address is specified outside the message text, as part of the SMTP exchange. This supports the use of "blind courtesy copy" (BCC) addressing. The other headers include the "date", the "subject", the "MIME-version", the "content-type", and the "content-transfer-encoding" headers. The body of the message is separated from the text of the message by at least one blank line.

From Hyatt1Babb@compulandus.com Tue May 22 03:03:15 2007
Return-Path: <Hyatt1Babb@compulandus.com>
Received: from 783C94B0 ([125.138.211.150])
by flax9.uwaterloo.ca (8.12.8/8.12.5) with SMTP id l4M73BqD009093
for <manager@speedy.uwaterloo.ca>; Tue, 22 May 2007 03:03:13 -0400
Message-Id: <200705220703.l4M73BqD009093@flax9.uwaterloo.ca>
From: "Hanna, Kyle" <Salgado3U@comfortsupply.com>
To: manager@flax9.uwaterloo.ca
Date: Tue, 22 May 2007 03:02:18 -0500
Subject: FWD: Debt Consolidation.
MIME-Version: 1.0
Content-Type: text/plain
Content-Transfer-Encoding: 7bit

Your approval is waiting

Refinancee with us 4 lower rate.

http://www.geocities.com/Sebastiancnb63747

The trouble with being punctual is that nobodys there to appreciate it.

Figure 2 Example E-mail Message Text

The LingPipe [14] library was used to parse the subject and the body of the text for content filtering purposes. The text was converted to lower-case characters and tokenized using white space and punctuation. Stop words, such as “a”, “an”, “the”, “and”, and “or”, were removed. Tokens were stemmed using Porter’s stemming algorithm [15]. The processed text for the example message is shown in the Figure 3 Processed Example E-mail Message Text.

subject : fwd : debt consolid .
your approv wait refinance us 4 lower rate . http : / / www . geociti . com /
sebastiancnb63747 troubl be punctual nobodi appreci .

Figure 3 Processed Example E-mail Message Text

The alpha-numeric tokens for each message are then converted to Term Frequency (TF) / Inverse Document Frequency (IDF) representation. The set of tokens observed in the training data are used to compute the TF/IDF value as shown in Equation 2.1 Term Frequency - Inverse Document Frequency, where i indexes the message and j indexes the token (aka term).

Equation 2.1 Term Frequency - Inverse Document Frequency

$$t_{i,j} = freq(term_j | message_i) \ln\left(\frac{count(messages)}{count(messages\ containing\ term_j)}\right)$$

The length of the TF/IDF vector for each message is scaled to one, by dividing the TF/IDF value for each term by the square root of the sum of squared TF/IDF values for the message. Scaling each vector in this manner reduces the effect of message length on the classification process.

2.4 Fraud Detection

Fraud is the intentional use of deception to gain some benefit, often financial gain. According to the most recent Federal Bureau of Investigation Financial Crimes Report to

the Public [16], there is an upward trend among many forms of financial crimes including health care fraud. Fraudulent billings to health care programs, both public and private, are estimated to be between 3 and 10 percent of total health care expenditures. This estimate is consistent with the most recent Association of Certified Fraud Examiners Report to the Nations [17], which showed survey participants estimated the typical organization loses 5% of its revenue each year.

Common types of fraud include tax fraud [18], securities fraud [19], health insurance fraud [20], auto insurance fraud [9] [21], credit card fraud [22], and telecommunications fraud [23] [24]. For tax fraud, a taxpayer intentionally avoids reporting income or overstates deductions. For securities fraud, a company misstates values on financial reports. For insurance fraud, the insured files claims that overstate losses. For credit card fraud, the credit card is used by someone other than the legitimate owner. Challenges for fraud detection include imbalanced class distributions, large data sets, class overlap, and the lack of publicly available data.

2.5 Insurance Claim Representation

Each auto insurance claim is represented by a vector of attribute values. This include both numeric and nominal attributes describing objects (vehicle and policy) as well as the claim event (accident area and the presence of witnesses). For example, the vehicle is described by the make (manufacturer), the age (in years), the price, and the

category (sports car, sedan, SUV, etc), while the insurance policy is described by the policy type (liability, collision, or comprehensive) and the deductible.

3. MACHINE LEARNING

Machine learning models are functions that map an input vector $\boldsymbol{x}_i$ to an output value y_i. A wide variety of machine learning algorithms are available, including classification (predicting nominal output values), regression (predicting numeric output values), and cluster analysis (predicting group membership). This chapter provides details of the model selection techniques, machine learning methods, and learning strategies used in this research.

3.1 Model Selection and Prediction

Model selection is the process of comparing the performance of various machine learning algorithms and parameterizations of the machine learning algorithms [25]. We want to choose the model with the lowest generalization error. A typical choice for measuring classification errors between the desired output y_i and the output estimated by a model $\hat{f}(\boldsymbol{x}_i)$ is shown in Equation 3.1 Zero-One Loss.

Equation 3.1 Zero-One Loss

$$Loss\left(y_i, \hat{f}(\boldsymbol{x}_i)\right) = I\left(y_i \neq \hat{f}(\boldsymbol{x}_i)\right) = \begin{cases} 0, & y_i = \hat{f}(\boldsymbol{x}_i) \\ 1, & y_i \neq \hat{f}(\boldsymbol{x}_i) \end{cases}$$

Consider the trade-off between error on the training data and complexity of the model shown in Figure 4 Error versus Complexity (Under versus Over Fitting). On the left side of the graph, our model appears to be too simple (high bias; under fitting). As the model grows more complex (more nodes are added to the decision tree), error on the training data is reduced but eventually error on the test data starts to increase. On the right side of the graph, our model appears to be too complex (high variance; over fitting). Model selection is about choosing the model which has best performance for the test data.

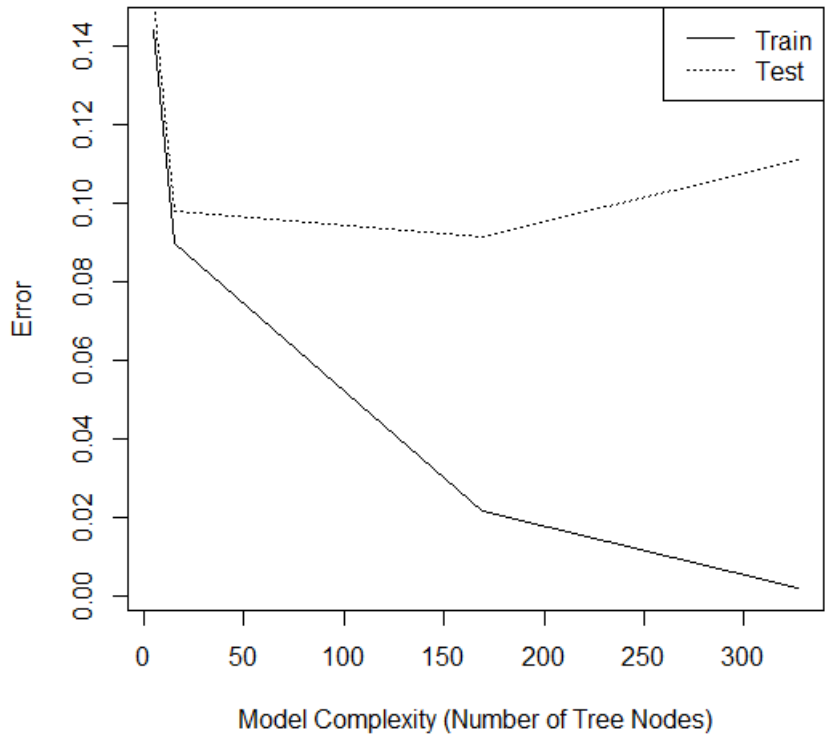

Figure 4 Error versus Complexity (Under versus Over Fitting)

The major model selection strategies include the following 5 evaluation methods:

1. Hold Out Validation Set Error: if annotated data is plentiful, the preferred method for model selection is to divide the data into disjoint training and validation sets. We choose the model which minimizes Equation 3.2 Expected Loss for the validation set. Once the learning method and the parameters have been selected, the combined training and validation sets are used for constructing the final model.

Equation 3.2 Expected Loss

$$\frac{\sum_{i=1}^{n} Loss\left(y_i, \hat{f}(\boldsymbol{x}_i)\right)}{n}$$

2. K-Fold Cross Validation Error: each of the observations in the training set is randomly assigned to one of K partitions of the data. We choose the model which minimizes Equation 3.3 Expected Cross Validation Loss for the training set, where $\hat{f}^{-K(i)}(\boldsymbol{x}_i)$ is the model constructed without the partition containing observation i. The partitioning scheme for 5-fold cross validation is shown in the Table 2 Partition Mapping for 5-Fold Cross Validation. The typical choices for K are 5 or 10. Smaller values for K often under fit the training data (over estimating the generalization error), while larger values for K often over fit the training data (under estimating the generalization error). With cross validation, it's important to note that only data used for the training partitions should be used to derive the model to be evaluated on the validation

partition. When the size of the annotated training data is small, this is the preferred method for model selection.

Equation 3.3 Expected Cross Validation Loss

$$\frac{\sum_{i=1}^{n} Loss\left(y_i, \hat{f}^{-K(i)}(\boldsymbol{x}_i)\right)}{n}$$

Table 2 Partition Mapping for 5-Fold Cross Validation

K=5	Partition 1	Partition 2	Partition 3	Partition 4	Partition 5
Model 1	Validation	Train	Train	Train	Train
Model 2	Train	Validation	Train	Train	Train
Model 3	Train	Train	Validation	Train	Train
Model 4	Train	Train	Train	Validation	Train
Model 5	Train	Train	Train	Train	Validation

3. Out of Bag (OOB) Error: a bootstrap sample of the training data is used to construct a model, and it is evaluated using observations that were not included in the bootstrap sample. A bootstrap sample of a training set of size n is a sample of size n, where sampling is performed with replacement. As n goes to ∞, the proportion of the training set included in the bootstrap sample is $\lim_{n\to\infty}\left(1-\left(1-\frac{1}{n}\right)^n\right) = 1-\exp(-1) \approx 63.2\%$. We choose the model which minimizes the Equation 3.4 Expected Out Of Bag (OOB) Loss using

the training data, where $b \in C^{-i}$ is a bootstrap sample that did not include observation i. This method can be useful for ensemble learning methods, where each of the ensemble members is constructed from a bootstrap sample of the data (e.g. Random Forests).

Equation 3.4 Expected Out Of Bag (OOB) Loss

$$\frac{\sum_{i=1}^{n} \frac{\sum_{b \in C^{-i}} Loss\left(y_i, \hat{f}^{*b}(\boldsymbol{x}_i)\right)}{|C^{-i}|}}{n}$$

4. Structural Risk Minimization: the training data and a measure of complexity of the model are used to estimate generalization error. Complexity is measured using the Vapnik-Chervonenkis (VC) dimensionality of the model [26]. The VC dimensionality of a model is the maximum number of observations that can be shattered by the model; i.e. that can separated no matter which of the 2^n possible labelings of the n training observations is observed. The Figure 5 Shattering of 3 Points by a 2-Dimensional Linear Classifier shows that 3 points can be completely shattered by a 2-dimensional linear classifier. In fact, it can be shown that $n + 1$ points can be shattered by an n-dimensional linear classifier. Equation 3.5 Upper Bound on Expected Loss shows the expected generalization error based on the training data and the complexity of the model, where h is the VC dimensionality of the model, and $1 - \eta$ is the probability of the true expected loss being less than the estimate.

Equation 3.5 Upper Bound on Expected Loss Using Structural Risk Minimization

$$\frac{\sum_{i=1}^{n} Loss\left(y_i, \hat{f}(\boldsymbol{x}_i)\right)}{n} + \frac{\varepsilon}{2}\left(1 + \sqrt{1 + \frac{4\sum_{i=1}^{n} Loss\left(y_i, \hat{f}(\boldsymbol{x}_i)\right)/n}{\varepsilon}}\right)$$

where $\varepsilon = 4\frac{h\left(ln\left((2n)/h\right)+1\right)-ln(\eta/4)}{n}$

In practice, this is a very loose (pessimistic) estimate of the expected loss, and it is difficult to estimate h for many models; so this method is not widely used in practice.

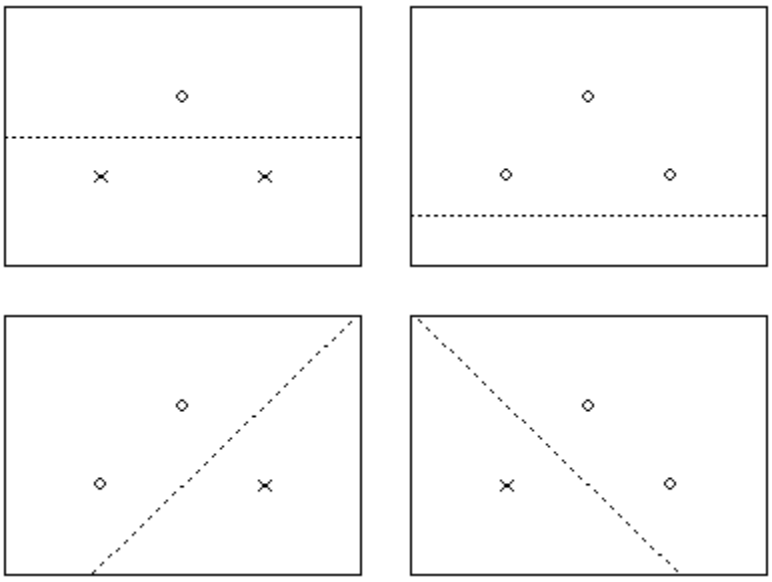

Figure 5 Shattering of 3 Points by a 2-Dimensional Linear Classifier

5. Empirical Risk Minimization: the training data is used to estimate Equation 3.2 Expected Loss. In practice, this estimate is often overly optimistic, as the

expected loss decreases as the trained model over fits the training data (with much worse performance experienced when generalizing to new data).

3.2 Taxonomy

A mapping of machine learning methods to thesis sections is shown in the Table 3 Taxonomy of Machine Learning Methods.

Table 3 Taxonomy of Machine Learning Methods

Method	Spam	Phishing	Fraud
Supervised Model: Decision Tree	5.1	6	7
Supervised Model: Support Vector Machine	5.4		
Supervised Ensemble: Logit Boost	5.2, 5.3		
Supervised Ensemble: Random Forest	5.1	6	7
Unsupervised Model: Partitioning Around Medoids	5.1		
Unsupervised Model: Spectral Clustering		6	
Unsupervised Model: Latent Dirichlet Allocation		6	
Unsupervised Model: One Class Support Vector Machine			7
Semi-Supervised Learning: Transduction		6	
Strategy: Active Learning	5.1		
Strategy: Adversarial Learning	5.4, 6		

3.3 Methods

The learning methods used in this research can be divided into three basic categories. Supervised learning methods require annotated data in order to map input data to annotated labels. Unlike supervised learning methods, unsupervised learning methods do not require annotated data. Instead unsupervised methods learn to group input data based on the learning method's objective function. Semi-supervised learning methods are a cross between supervised and unsupervised learning methods. Semi-supervised learning methods incorporate both annotated and unannotated data into the learning process.

3.3.1 Supervised Learning

In the supervised learning setting, a set of tuples $\boldsymbol{D} = \{(\boldsymbol{x}_n, y_n)\}_{i=1}^{n} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_n, y_n)\}$ mapping input vector $\boldsymbol{x}_i$ to output value y_i is provided to the learning algorithm to construct the mapping function: $\hat{f}(\boldsymbol{x}_i) \rightarrow \hat{y}_i$. The circumflex (^) is placed above the y_i to indicate the output of the function is a prediction made by a machine learning model instead of a label assigned by an annotator. As George Box said, "All models are wrong, but some are useful." It is called supervised learning, because the target output y_i values are provided to the learning algorithm. In regression the y_i value is numeric, while in classification the y_i value is nominal. For spam detection, phishing detection, and fraud detection, the y_i values are nominal: $y_i \in \{Spam, NotSpam\}$ for spam detection, $y_i \in \{Phishing, NotPhishing\}$ for phishing detection, and $y_i \in \{Fraud, NotFraud\}$ for fraud detection. For binary classification problems, such as

spam, phishing and fraud detection, the nominal class value is often encoded as a numeric value: $y_i = +1$ indicating an observation belongs to the positive class (spam, phishing, or fraud) and $y_i = -1$ indicating an observation belongs to the negative class. The elements of the $\boldsymbol{x}_i$ vectors can be either numeric or nominal values used to predict output value $\hat{y}_i$.

3.3.1.1 Decision Trees

The Classification And Regression Trees (CART) algorithm [27] uses the Equation 3.6 Gini Impurity Measure to recursively partition a training set, forming a decision tree.

Equation 3.6 Gini Impurity Measure

$$Gini(\boldsymbol{D}) = \sum_{i=1}^{k} \Big(Prob(Class_i) * \big(1 - Prob(Class_i)\big) \Big) = 1 - \sum_{i=1}^{k} Prob(Class_i)^2$$

For binary (two class) classification problems, such as spam detection, phishing detection, and fraud detection, the Gini impurity measure can be expressed as a function of the proportion of the training set belonging to the "positive" class (spam for spam detection, phishing for phishing detection, or fraud for fraud detection). As shown in the Figure 6 Graph of the Gini Impurity Function, the Gini impurity function returns the largest value when the set contains an equal number of observations from the positive class and the negative class. The Gini impurity function returns the smallest value (zero) when the set contains only members of one of the classes.

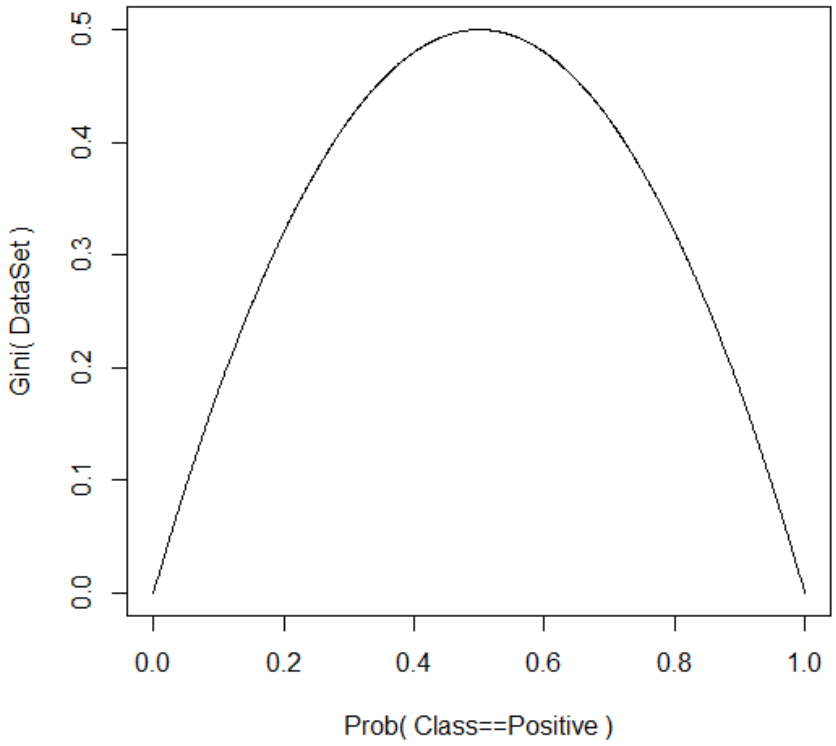

Figure 6 Graph of the Gini Impurity Function

Decision tree construction is known as recursive partitioning, as the decision tree learning algorithm chooses the condition that provides the greatest reduction in impurity for each node in the tree, beginning with the entire training set in the root node. The algorithm is shown in the Figure 7 Decision Tree Partitioning Algorithm.

```
BestSplitCondition = unassigned
BestGiniDecrease = - Infinity
for each feature f
   d = number of distinct values for feature f
   FeatureClassCounts = allocate d x 2 matrix to hold class counts
   TotalClassCounts = allocate array of 2 class counts
   for i = 1 to n
      increment FeatureClassCounts[FeatureIndex(x[i,f]), ClassIndex(y[i])]
      increment TotalClassCounts[ClassIndex(y[i])]
   if IsNumeric[f]
      CumulativeCounts = array of 2 class counts
      for i = 1 to d
        increment CumulativeClassCounts using row i of FeatureClassCounts
        current = GiniDecrease(CumulativeClassCounts, TotalClassCounts)
        if current > BestGiniDecrease
           BestGiniDecrease = current
           BestSplitCondition = (f, i)
   else
      for i = 1 to d
        current = GiniDecrease(FeatureClassCounts for row i, TotalClassCounts)
        if current > BestGiniDecrease
           BestGiniDecrease = current
           BestSplitCondition = (f, i)
```

Figure 7 Decision Tree Partitioning Algorithm

Consider the Table 4 Example Data Points. The split point **X**[,1] = 3.93 (the mean of 2.63 and 5.23) reduces Gini impurity from 0.5 to 0.0, while the split point **X**[,1] = 2.09 (the mean of 1.82 and 2.36) only reduces Gini impurity from 0.5 to 0.4. Therefore **X**[,1] = 3.93 would be preferred to **X**[,1] = 2.09.

Table 4 Example Data Points

X[,1]	**X**[,2]	y
1.82	7.72	+1
2.36	6.26	+1
2.63	6.97	+1
5.23	2.92	-1
6.17	2.77	-1
6.82	1.98	-1

The complexity of the decision tree partitioning algorithm is shown in Equation 3.7 Runtime Complexity of the Decision Tree Partitioning Algorithm, where f is the number of features and n is the number of training examples (with n being greater than or equal to d).

Equation 3.7 Runtime Complexity of the Decision Tree Partitioning Algorithm

$$O(f * n)$$

3.3.1.2 Support Vector Machines

A Support Vector Machine (SVM) is a binary classification model comprised of a maximum margin hyperplane. The "support vectors" are training set observations that define the hyperplane. The margin is the distance between the support vectors and the hyperplane. The SVM is defined by solving the following optimization problem:

Equation 3.8 Primal Support Vector Machine Optimization Problem

$$\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{n}\xi_i$$

subject to $y_i(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_i) + b) \geq 1 - \xi_i$ and $\xi_i \geq 0\ \forall i \in \{i\}_{i=1}^{n}$

The ξ_i are hinge loss values; i.e. $\xi_i = 1 - max\big(0, 1 - y_i(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_i) + b)\big)$. The C is a cost parameter for L1 regularization of the loss values, to balance model complexity and training set performance. The values of C range from a value small enough to use all training observations as support vectors (possibly over fitting the training data) to a value large enough to use a minimum number of training observations as support vectors (possibly under fitting the training data).

Rather than explicitly computing the function $\boldsymbol{\phi}(\boldsymbol{x}_i)$, the equivalent dual optimization problem is solved:

Equation 3.9 Dual Support Vector Machine Optimization Problem

$$\min_{\boldsymbol{\alpha}} \frac{1}{2}\boldsymbol{\alpha}^T \boldsymbol{Q}\boldsymbol{\alpha} - \boldsymbol{e}^T\boldsymbol{\alpha}$$

subject to $\boldsymbol{y}^T\boldsymbol{\alpha} = 0$ and $0 \leq \alpha_i \leq C \ \forall i \in \{i\}_{i=1}^{n}$

where $Q_{i,j} = Kernel(\boldsymbol{x}_i, \boldsymbol{x}_j)$

The scalar α_i values are known as Lagrange multipliers. A vector $\boldsymbol{\alpha}$ is a solution of the optimization problem if the Karush-Kuhn-Tucker (KKT) conditions are met for all i:

Equation 3.10 Karush Kuhn Tucker (KKT) Optimization Conditions

$$\alpha_i = 0 \implies y_i\left(\sum_{j=1}^{n} y_j\alpha_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) + \rho\right) > 1$$

$$0 < \alpha_i < C = 0 \implies y_i\left(\sum_{j=1}^{n} y_j\alpha_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) + \rho\right) = 1$$

$$\alpha_i = C \implies y_i\left(\sum_{j=1}^{n} y_j\alpha_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) + \rho\right) < 1$$

The Sequential Minimal Optimization (SMO) algorithm [28] is the most commonly used method to solve this optimization problem. Rather than attempting to solve the optimization problem for all observations simultaneously (or a large "chunk" of the observations), the SMO algorithm loops through the training set observations and solves the optimization for only a pair of observations at a time. The working set

(observation pair) selection heuristics vary among SMO implementations. For example, the working set selection heuristics for the LIBSVM SVM library [29] are described in [30]. While the actual runtime varies among implementations, the runtime is typically somewhere between linear and quadratic in the number of observations.

The SMO algorithm is briefly sketched in the Figure 8 Sequential Minimal Optimization Algorithm, with the actual optimization operation shown in the Figure 9 Lagrange Multiplier Optimization for the SMO Algorithm.

while at least one Lagrange multiplier violates the KKT conditions
  select an α_1 that violates the KKT conditions
  heuristically select another Lagrange multiplier α_2
  optimize α_1 and α_2

Figure 8 Sequential Minimal Optimization Algorithm

if y_1 != y_2

$L = max(0, a_2^{old} - a_1^{old})$

$H = min(C, C + a_2^{old} - a_1^{old})$

else

$L = max(0, a_1^{old} + a_2^{old} - C)$

$H = min(C, a_1^{old} + a_2^{old})$

$$\eta = 2\,K(\boldsymbol{x}_1, \boldsymbol{x}_2) - K(\boldsymbol{x}_1, \boldsymbol{x}_1) - K(\boldsymbol{x}_2, \boldsymbol{x}_2)$$

$$\alpha_2^{new} = \alpha_2^{old} - \frac{y_2\left(\left(\hat{f}(\boldsymbol{x}_1) - y_1\right) - \left(\hat{f}(\boldsymbol{x}_2) - y_2\right)\right)}{\eta}$$

$$\alpha_2^{new} = \begin{cases} H, & \alpha_2^{new} \geq H \\ \alpha_2^{new}, & L < \alpha_2^{new} < H \\ L, & \alpha_2^{new} \leq L \end{cases}$$

$$\alpha_1^{new} = \alpha_1^{old} + y_1\, y_2\, \left(\alpha_2^{old} - \alpha_2^{new}\right)$$

Figure 9 Lagrange Multiplier Optimization for the SMO Algorithm

For example, consider again the 6 observations shown in the Table 4 Example Data Points. The decision boundary for an SVM trained on this data using a linear dot product kernel is shown in the Figure 10 Example SVM Decision Boundary. There are two support vectors $\alpha_2 = \alpha_4 = 0.103$, with $\rho = 0.458$. Since the decision boundary is a linear hyperplane in the input space, we can derive the linear model as $\boldsymbol{w} = \left[\sum_{i=1}^{n} y_i \alpha_i X_{i,1}, \sum_{i=1}^{n} y_i \alpha_i X_{i,2}\right] = [0.296, -0.344]$ with $b = 0.458$. The weight vector is shown by the solid arrow. By solving for $\boldsymbol{w}^T \boldsymbol{x} + \rho = 0$, we find the decision boundary is defined by the solid line $X_{i,2} = 0.859\, X_{i,1} + 1.329$. The projections of the support

vectors onto the decision boundary are shown by the dotted arrows. The width of the margin is $\frac{1}{\sqrt{(0.296)^2+(-0.344)^2}}$, with the margins identified by the dashed lines.

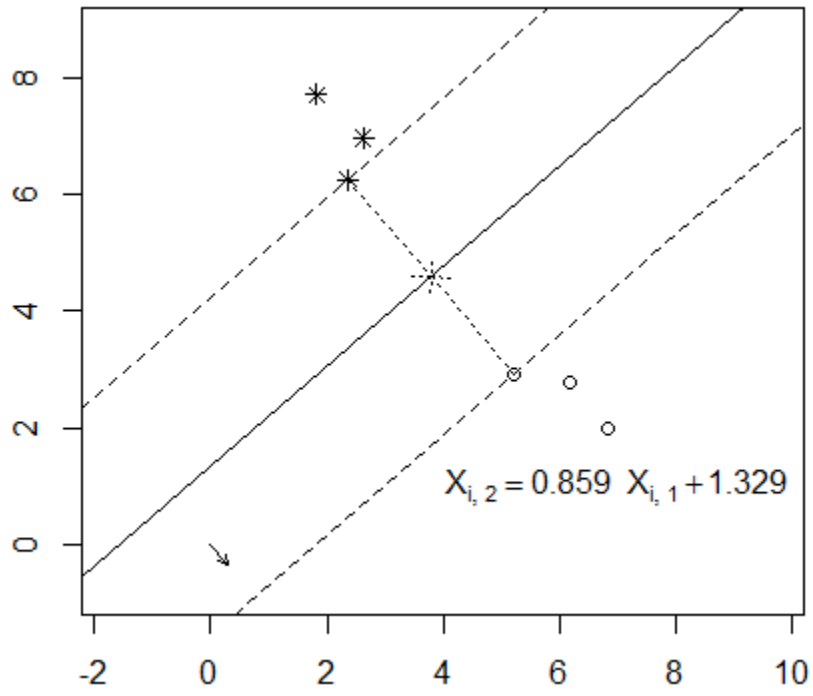

Figure 10 Example SVM Decision Boundary

A vector of weights can be used as a simple linear binary classifier. If the sum of a bias term and the dot product of a new observation with the weight vector is positive, the new observation is classified as a member of the positive class; otherwise, the new observation is classified as a member of the negative class. The decision function for the SVM shown in Figure 10 Example SVM Decision Boundary is $sgn(\sum_{i=1}^{n} y_i \alpha_i K(\boldsymbol{x}_i, \boldsymbol{x}_{new}) + \rho) = sgn(\boldsymbol{w}^T \boldsymbol{x} + b)$.

SVMs can learn non-linear decision boundaries via the "kernel trick." The output of a kernel in the original input space is equivalent to a dot product for a higher dimensional feature space if Equation 3.11 Mercer's Kernel Condition is satisfied.

Equation 3.11 Mercer's Kernel Condition

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\left(c_i c_j Kernel\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right)\right) \geq 0$$

Commonly used kernels include the Equation 3.12 Linear Dot Product Kernel, the Equation 3.13 Non-Linear Polynomial Kernel, and the Equation 3.14 Non-Linear Radial Basis Function (Gaussian) Kernel.

Equation 3.12 Linear Dot Product Kernel

$$Kernel\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) = \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle = \boldsymbol{x}_i{}^T \boldsymbol{x}_j$$

Equation 3.13 Non-Linear Polynomial Kernel

$$Kernel\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) = \left(scale * \left(\boldsymbol{x}_i{}^T \boldsymbol{x}_j\right) + offset\right)^{degree}$$

Equation 3.14 Non-Linear Radial Basis Function (Gaussian) Kernel

$$Kernel\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) = exp\left(-\frac{\left\|\boldsymbol{x}_i - \boldsymbol{x}_j\right\|^2}{\sigma}\right)$$

3.3.1.3 Ensemble Methods

An ensemble is a group of machine learning models. Commonly used ensembles include Gradient Boosting Machines (GBMs), such as Logit Boost, and Random Forests.

3.3.1.3.1 Logit Boost

Logit Boost is also known as additive logistic regression. It is a form of logistic regression where the residual error is used to update the target response for the classifier. Consider the standard logistic regression problem. The logistic function maps a log odds value v to a probability p, as illustrated in the Equation 3.15 Logistic Function and the Figure 11 Graph of the Logistic Function.

Equation 3.15 Logistic Function

$$p = \frac{1}{1 + exp(-v)}$$

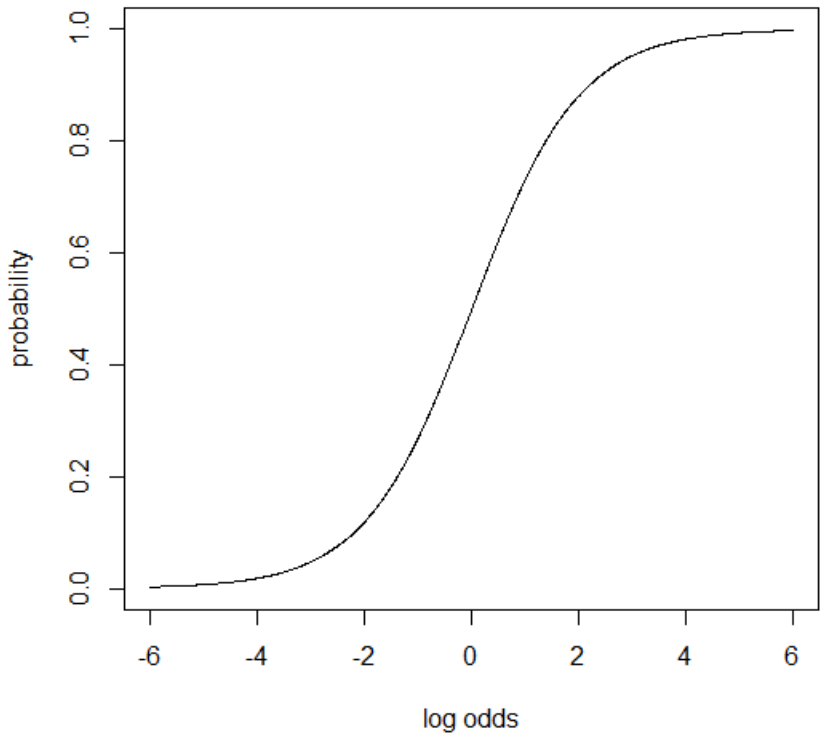

Figure 11 Graph of the Logistic Function

Given a data set $D = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n}$ where $\boldsymbol{x}_i$ is a feature vector and $y_i \in \{-1, +1\}$, the goal of logistic regression is to find a weight vector $\boldsymbol{w}$ such that $\frac{1}{1+\exp(-\boldsymbol{w}^T \boldsymbol{x}_i)} \approx {y_i}^*$. The formula for ${y_i}^*$ is shown in the Equation 3.16 Conversion from Positive/Negative Outcome to Binary (One/Zero).

Equation 3.16 Conversion from Positive/Negative Outcome to Binary (One/Zero)

$$ {y_i}^* = \frac{y_i + 1}{2} $$

This is a simple linear model where we predict $\hat{y}_i = +1$ if the $\boldsymbol{w}^T \boldsymbol{x}_i$ is greater than zero and $\hat{y}_i = -1$ otherwise. A number of learning algorithms exist for logistic regression, with stochastic gradient descent being an example that scales well to large data sets. The weight vector is initialized with small random values and updated using the negative gradient of the logistic loss function. The logistic loss function definition is shown in the Equation 3.17 Negative Log Likelihood (Logistic) Loss Function.

Equation 3.17 Negative Log Likelihood (Logistic) Loss Function

$$ Loss\big(\hat{f}(x_i), y_i\big) = -log\left(\frac{1}{1 + exp(-y_i\, \boldsymbol{w}\, \boldsymbol{x}_i)}\right) = log\big(1 + exp(-y_i\, \boldsymbol{w}\, \boldsymbol{x}_i)\big) $$

$$ = -\Big({y_i}^*\ \log \hat{f}(\boldsymbol{x}_i) + (1 - {y_i}^*) \log\Big(1 - \hat{f}(\boldsymbol{x}_i)\Big)\Big) $$

To optimize (minimize) the loss function, we can add the negative gradient to the weight vector. The partial derivative of the loss function with respect to the weight

vector $\boldsymbol{w}$ is shown in the Equation 3.18 Negative Gradient of the Negative Log Likelihood Loss Function.

Equation 3.18 Negative Gradient of the Negative Log Likelihood Loss Function

$$-\frac{\partial Loss\left(\hat{f}(\boldsymbol{x}_i), y_i\right)}{\partial \boldsymbol{w}} = y_i\, \boldsymbol{x}_i\, \frac{1}{1 + exp(y_i\, \boldsymbol{w}\, \boldsymbol{x_i})} = \left(y_i{}^* - \hat{f}(\boldsymbol{x_i})\right)\, \boldsymbol{x}_i$$

Adding the negative gradient to the weight vector for a randomly selected observation is known as stochastic gradient descent. The weight update step shown in the Equation 3.19 Stochastic Gradient Descent Weight Update reduces the value of the loss function for the training observation.

Equation 3.19 Stochastic Gradient Descent Weight Update

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \frac{\partial Loss\left(\hat{f}(\boldsymbol{x}_i), y_i\right)}{\partial \boldsymbol{w}}$$

Unlike the stochastic gradient descent approach to logistic regression, additive logistic regression (Logit Boost) automatically discretizes numeric values and selects features to be incorporated into the model. Each model in the ensemble modifies the log odds of an observation belonging to the positive class, ideally increasing the log odds for members of the positive class and decreasing the log odds for members of the negative class.

The Logit Boost algorithm [31] is shown in the Figure 12 Logit Boost Learning Algorithm. The Logit Boost algorithm can be described as a form of gradient boosting

machine [32], with a logistic loss function. Performance is boosted (improved) by iteratively using the Newton-Raphson algorithm (a Taylor Series approximation) to estimate the residual error and weighted regression to reduce the error. Weights are based on the uncertainty of the prediction. The closer the log odds are to zero, the more uncertain the model is about the prediction.

$\forall i\colon\ w_i = \frac{1}{n}, p_i = \frac{1}{2}$

for m = 1 to M

$\forall i\colon\quad r_i = \frac{y_i{}^* - p_i}{p_i(1-p_i)}, \quad w_i = p_i(1-p_i)$

fit $\hat{f}_m(x)$ as a weighted least squares regression of r_i to x_i using weights w_i

$\forall i\colon\quad p_i = \frac{1}{1+\exp\left(-\sum_{j=1}^{m} \hat{f}_j(\boldsymbol{x}_i)\right)}$

output classifier $F(\boldsymbol{x}_i) = sgn\left(\sum_{m=1}^{M} \hat{f}_m(\boldsymbol{x}_i)\right)$

Figure 12 Logit Boost Learning Algorithm

A common method for fitting $\hat{f}_m(\boldsymbol{x})$ is the use of regression stumps. The learning method for regression stumps is shown in the Figure 13 Regression Stump Learning Algorithm.

```
BestSplitCondition = unassigned
BestVarianceDecrease = - Infinity
for each feature f
   d = number of distinct values for feature f
   FeatureResponses = allocate d x 3 array to hold weights, weighted responses,
      and weighted squared responses
   TotalResponses = allocate array of 3 values to hold weights, weighted responses,
      and weighted squared responses
   for i = 1 to n
      add weight, weighted response, and weighted squared response to
         FeatureResponses[FeatureIndex(x[i,f]),]
      add weight, weighted response, and weighted squared response to TotalResponses
   if IsNumeric[f]
      CumulativeResponses = array of 3 values to hold weights, weighted responses,
         and weighted squared responses
      for i = 1 to d
        increment CumulativeResponses using row i of FeatureResponses
        current = VarianceDecrease(CumulativeResponses, TotalResponses)
        if current > BestVarianceDecrease
           BestVarianceDecrease = current
           BestSplitCondition = (f, i)
   else
      for i = 1 to d
        current = VarianceDecrease(FeatureResponses for row i, TotalResponses)
        if current > BestVarianceDecrease
           BestVarianceDecrease = current
           BestSplitCondition = (f, i)
```

Figure 13 Regression Stump Learning Algorithm

The sum of weights, sum of weighted responses, and sum of weighted squared responses are sufficient to compute the weighted variance as shown in Equation 3.20 Variance for Weighted Responses; because the variance is simply the difference between the expected value of the squared response and the squared expected response for the weighted observations. Ideally, all observations on the same side of the split would have the same value (variance of zero).

Equation 3.20 Variance for Weighted Responses

$$\hat{\sigma}^2 = \frac{\sum_{i=1}^{n} w_i (r_i)^2}{\sum_{i=1}^{n} w_i} - \left(\frac{\sum_{i=1}^{n} w_i r_i}{\sum_{i=1}^{n} w_i} \right)^2$$

For numeric values, the two possible partitions are “less than or equal to the split value” and “greater than the split value”, while for nominal values, the two possible partitions are “equal to the value” (or within a set of values) and “not equal to the value” (or not within a set of values). An added benefit of the Logit Boost approach is that it makes it easy to handle missing values. For example, observations with missing values for a particular feature can simply be treated as a third partition for the regression stump, with the mean response used if there are no missing values for that particular feature.

The runtime complexity of the regression stump algorithm is the same as the runtime complexity for the partitioning algorithm for a decision tree node.

3.3.1.3.2 Random Forest

Both adaptive resampling and model combination (arcing) and bootstrap aggregation (bagging) have been shown to reduce the variance (complexity) of a classifier [33] [34]. A Random Forest classification model is an ensemble of decision trees, constructed by the algorithm shown in the Figure 14 Random Forest Learning Algorithm. The use of bootstrap sampling helps to reduce the variance (complexity) of the ensemble classifier by training the trees on different subsets of the data, while the use of random subspace projections (random feature selection) for each node mitigates the tendency of the greedy decision tree optimization algorithm to over fit the training data.

for m = 1 to M

Draw a bootstrap sample $\boldsymbol{D}^*$ of size n from the training data

Grow a tree $\hat{f}_m(\boldsymbol{x}_i)$, by recursively partitioning the data as follows:

Select p features at random from the input features

Partition the data according to Figure 7 Decision Tree Partitioning Algorithm

Output classifier $\hat{F}(\boldsymbol{x}_i) = MajorityVote\left(\{\hat{f}_m(\boldsymbol{x}_i)\}_{m=1}^{M}\right)$

Figure 14 Random Forest Learning Algorithm

The output of a random forest classifier is simply the probability that a randomly selected tree in the ensemble would assign a particular class label, but this probability is interpreted as the probability of the observation belonging to that particular class.

3.3.2 Unsupervised Learning

In the unsupervised learning setting, only the $\boldsymbol{x}_i$ input vectors are provided to the learning algorithm to construct the mapping function: $\hat{f}(\boldsymbol{x}_i) \rightarrow \hat{y}_i$. Partitioning Around Medoids (PAM), spectral clustering, Latent Dirichlet Allocation (LDA), and one class Support Vector Machines (SVMs) were used as part of this research.

3.3.2.1 Partitioning Around Medoids

The Partitioning Around Medoids (PAM) algorithm is used for clustering (grouping) similar observations together. There are two major steps to the PAM algorithm: the build step (initialization), followed by the swap step [a variant of the Expectation-Maximization (EM) algorithm]. The build step is shown in the Figure 15 Partitioning Around Medoids Initialization. The runtime complexity of the build step is quadratic in the number of training set observations.

for i = 1 to $n - 1$

  for j = 2 to n

    $Distance_{i,j} = Distance_{j,i} = \|\boldsymbol{x}_i - \boldsymbol{x}_j\| = distance(\boldsymbol{x}_i, \boldsymbol{x}_j)$

$\boldsymbol{Medoids}_1 = \boldsymbol{x}_{\min_i \sum_{j=1}^{n} Distance_{i,j}}$

for i = 1 to n

  $DistanceToNearestMedoid_i = \|\boldsymbol{x}_i - \boldsymbol{Medoids}_1\|$

for k = 2 to K

  for i = 1 to n

    If $x_i \notin \boldsymbol{Medoids}$

      $gain_i = 0$

      For j = 1 to n

        If $(j \neq i)\ and\ (\boldsymbol{x}_j \notin \boldsymbol{Medoids})$

          If $DistanceToNearestMedoid_j > Distance_{i,j}$

            $gain_i = gain_i + (DistanceToNearestMedoid_j - Distance_{i,j})$

  $\boldsymbol{Medoids}_k = \boldsymbol{x}_{\max_i gain_i}$

  for i = 1 to n

    $DistanceToNearestMedoid_i = \min_j \|\boldsymbol{x}_i - \boldsymbol{Medoids}_j\|$

Figure 15 Partitioning Around Medoids Initialization Algorithm

The parameter for the Partitioning Around Medoids (PAM) algorithm is the number of clusters. This can be set by executing the algorithm for multiple values for the number of clusters and selecting the model which maximizes the average silhouette value. The silhouette value for each observation is computed as shown in the Equation 3.21 Silhouette Value.

Equation 3.21 Silhouette Value

$$s_i = \frac{a_i - b_i}{max(a_i, b_i)}$$

where …

$$a_i \equiv \frac{\sum_{\boldsymbol{x}_j \in AssignedCluster} distance(\boldsymbol{x}_i, \boldsymbol{x}_j)}{|AssignedCluster|}$$

$$b_i \equiv \frac{\sum_{\boldsymbol{x}_j \in NeighborCluster} distance(\boldsymbol{x}_i, \boldsymbol{x}_j)}{|NeighborCluster|}$$

The silhouette value incorporates both a measure for cluster compactness, the average distance to other observations in the assigned cluster, and a measure of cluster separation, the average distance to observations in the nearest neighboring cluster. Silhouette values range between -1, indicating the observation does not fit well with the assigned cluster, and +1, indicating the observations fits well with the assigned cluster. Larger silhouette values indicate a better quality of clustering; i.e., a larger separation between the assigned cluster and the nearest alternative cluster, and/or a more compact assigned cluster.

The swap step is shown in the Figure 16 Partitioning Around Medoids Expectation Maximization Algorithm. The runtime complexity of the swap step is quadratic in the size of the largest cluster.

For i = 1 to n

$Cluster_i = \min_j \left\| \boldsymbol{x}_i, \boldsymbol{Medoids}_j \right\|$

For k = 1 to K

$for\ i\ \in Cluster_k$

$ClusterDistanceSum_i = 0$

$for\ j\ \in Cluster_k, i \neq j$

$ClusterDistanceSum_i = ClusterDistanceSum_i + Distance_{i,j}$

$\boldsymbol{Medoids}_k = \boldsymbol{x}_{\min_i ClusterDistanceSum_i}$

Figure 16 Partitioning Around Medoids Expectation Maximization Algorithm

There are a couple of distinct advantages of the PAM algorithm compared to the computationally cheaper K-means clustering algorithm. First, the initialization step is heuristically chosen based on the objective function (minimization of the sum of distances to the nearest medoid), instead of random initialization. Second, each cluster is represented by an actual training set observation instead of a mean vector.

3.3.2.2 Spectral Clustering

The advantage of spectral clustering is the ability for clusters to have arbitrary shapes. For example, neither K-means nor the PAM algorithm would be able to detect the two "clusters" shown in the "red" and "black" clusters shown in the Figure 17 Example of Spiral Clusters.

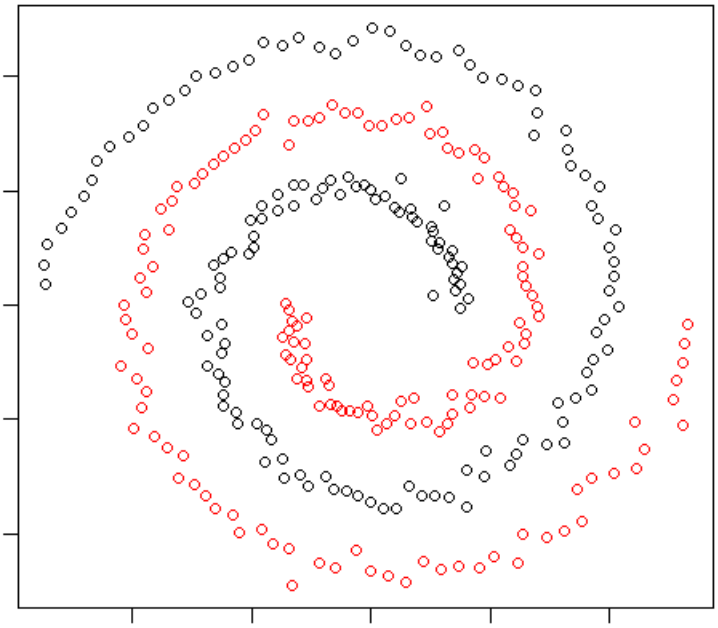

Figure 17 Example of Spiral Clusters

There are six basic steps in spectral clustering:

1. An affinity matrix is computed, where $\boldsymbol{A}[i,j] = \exp\left(-\frac{distance(i,j)^2}{2\sigma^2}\right)$ for i ≠ j and 0 otherwise. The affinity matrix for the observations in the Table 4 Example Data Points (with σ = 1) appears in the Table 5 Example Affinity Matrix.

Table 5 Example Affinity Matrix

	0.59	0.18			
0.59		0.64			
0.18	0.64				
				0.75	0.54
			0.75		0.30
			0.54	0.30	

The non-zero affinity values only appear where observations are in close proximity to one another.

2. A degree matrix is computed, where $\boldsymbol{D}[i,j] = \sum_{i=1}^{n} \boldsymbol{A}[i,j]$ for i = j and 0 otherwise. The degree matrix for the affinity matrix shown in the Table 5 Example Affinity Matrix is shown in the Table 6 Example Degree Matrix.

Table 6 Example Degree Matrix

0.77					
	1.23				
		0.82			
			1.29		
				1.05	
					0.84

If an observation is similar to many other observations, it will have a larger degree value.

3. A normalized Laplacian matrix is computed, where $\boldsymbol{L} = \boldsymbol{D}^{-1/2} * \boldsymbol{A} * \boldsymbol{D}^{-1/2}$. The Laplacian matrix for the degree matrix shown in the Table 5 Example Affinity Matrix is shown in the Table 7 Example Laplacian Matrix.

Table 7 Example Laplacian Matrix

	0.61	0.23			
0.61		0.63			
0.23	0.63				
				0.64	0.52
			0.64		0.32
			0.52	0.32	

Normalization discounts affinity values associated with observations having larger degree values.

4. The spectral decomposition of the Laplacian matrix is computed, deriving the eigenvalues and eigenvectors of the Laplacian matrix. The eigenvectors identify the characteristic vectors of the Laplacian matrix; i.e. the vectors that are simply scaled when multiplied by the Laplacian matrix. The eigenvalues indicate the associated scale factor. For the Table 7 Example Laplacian

Matrix, the eigenvalues are shown in the Table 8 Example Eigenvalues and the eigenvectors are shown in the Table 9 Example Eigenvectors.

Table 8 Example Eigenvalues

1.00	0.99995	-0.23	-0.31	-0.69	-0.77

Table 9 Example Eigenvectors

-0.36	0.38	0.72			0.45
-0.45	0.48	-0.01			-0.75
-0.37	0.39	-0.69			0.48
-0.46	-0.44		-0.11	-0.76	
-0.42	-0.39		-0.59	0.56	
-0.37	-0.35		0.80	0.32	

The eigenvectors associated with the positive eigenvalues can now be used as a spectral representation for the observations. The spectral representation for the observations is plotted in the Figure 18 Example Spectral Representation.

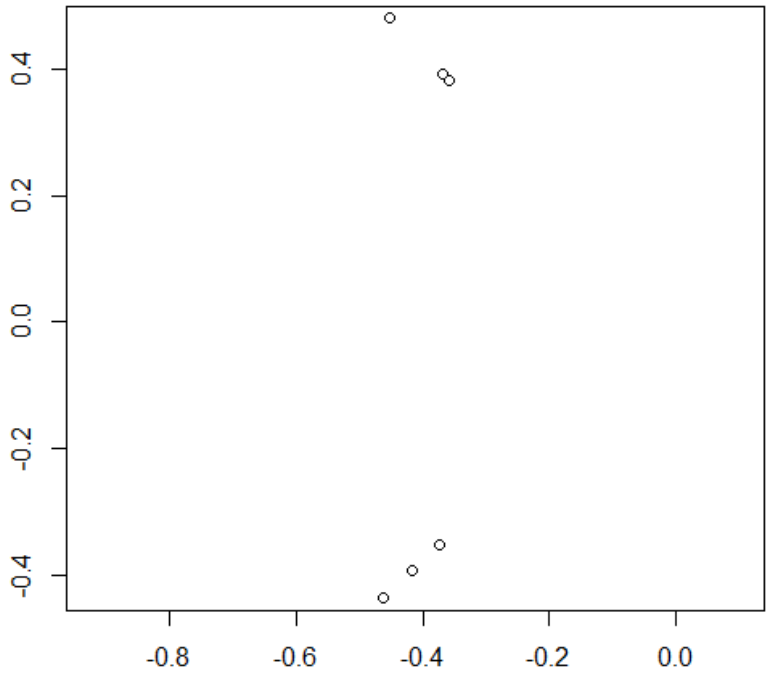

Figure 18 Example Spectral Representation

5. The spectral representations for the observations are scaled to unit length (by dividing each cell by the square root of the sum of squared values for the associated row).

6. The spectral representations for the observations are then clustered using an algorithm such as Partitioning Around Medoids (PAM).

3.3.2.3 Latent Dirichlet Allocation

Latent Dirichlet Allocation (LDA) is a generative model that can be used to discover topics (groups of related terms) in a collection of documents and assign topic

probabilities to text documents. LDA assumes the generative model illustrated in the Figure 19 Graphical LDA Generative Model.

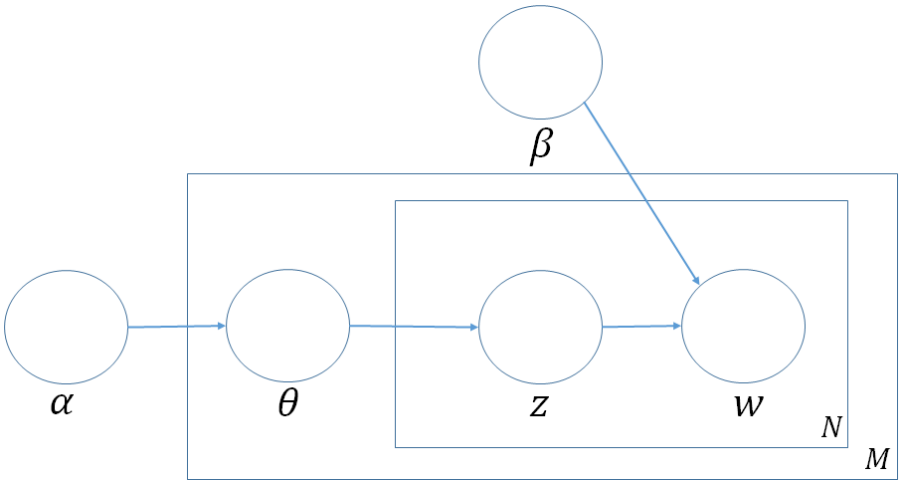

Figure 19 Graphical LDA Generative Model

The generative process is shown in the Figure 20 LDA Document Generation Process.

1. Choose $N \sim Poisson(\xi)$
2. Choose $\boldsymbol{\theta} \sim Dirichlet(\boldsymbol{\alpha})$
3. For each of the N words w_n
    a. Choose a topic $z_n \sim Multinomial(\boldsymbol{\theta})$
    b. Choose a word w_n from $p(w_n|z_n, \boldsymbol{\beta})$, a multinomial probability conditioned on the topic and the word probabilities for that topic

Figure 20 LDA Document Generation Process

Given a collection of documents, the goal of an LDA learning algorithm is to learn a set of topics. The learned collection of topics can then be used to estimate topic probabilities for new documents that are believed to have been generated using the same model. Collapsed Gibbs sampling is used to estimate the word probabilities and the topic probabilities for a document collection. A Gibbs sampler is a Markov Chain Monte Carlo (MCMC) algorithm that generates a sequence of samples based of the joint distribution of random variables. Instead of estimating the parameters of interest directly from a training corpus, word-to-topic assignments are used to derive the parameters of interest. A random initialization of word-to-topic assignments can be used to estimate the probability of each word for a topic as well as the probability of each topic for each document. These probabilities can then be used to iteratively update the word-to-topic assignments by randomly selecting each word-to-topic assignment based on the joint probability $p(topic|document) * p(word|topic)$. These updates are repeated until convergence or a set number of iterations has been completed. The runtime complexity

of this algorithm is equal to the product of the number of words and the number of topics, for each iteration.

3.3.2.4 One Class Support Vector Machines

One class Support Vector Machines (SVMs) [35] are used to determine how similar a new observation appears to be to be to a previously observed group of observations. The same Sequential Minimal Optimization (SMO) algorithm, used to derive the Lagrange coefficients for a binary classification Support Vector Machine (SVM), is used to derive the Lagrange coefficients for a one class SVM. The dual optimization problem is shown in Equation 3.22 Dual Optimization Problem for One Class SVM.

Equation 3.22 Dual Optimization Problem for One Class SVM

$$\min_{\boldsymbol{\alpha}} \frac{1}{2} \boldsymbol{\alpha}^T \boldsymbol{Q} \boldsymbol{\alpha}$$

subject to $\sum_{i=1}^{n} \alpha = \nu n$ and $0 \leq \alpha_i \leq 1 \; \forall i \in \{i\}_{i=1}^{n}$

where $\boldsymbol{Q}_{i,j} = Kernel(\boldsymbol{x}_i, \boldsymbol{x}_j)$

The non-zero α_i identify support vectors; i.e. the training set observations that identify the boundary surrounding the training set observations. The ν parameter places an upper bound on the proportion of training set observations that can be declared to be outliers, as well as a lower bound on the proportion of training set observations that are declared as support vectors.

3.3.3 Semi-Supervised Learning

Semi-supervised learning incorporates labeled (annotated) and unlabeled data into the learning process [36]. Semi-supervised learning methods rely on three basic assumptions [37]:

1. The Cluster Assumption: examples that group together belong to the same class.
2. The Smoothness Assumption (a corollary to the cluster assumption): a decision boundary will lie in a low density area rather than partitioning a high-density area.
3. The Manifold Assumption: a decision boundary in a high-dimensional feature space can be effectively represented on a lower dimensional boundary.

Self-training is a popular form of semi-supervised learning [38]. For example, an initial set of annotated data is used to train a classifier; the new classifier is used to assign labels to a set of unannotated data; then the initial and newly labeled data are used to train a final classifier.

Transduction is an alternative form of semi-supervised learning that has been incorporated into this research. Figure 21 Transductive versus Inductive Inference compares transductive inference to inductive inference [39]. Transductive inference incorporates the observations to be evaluated as part of the learning process.

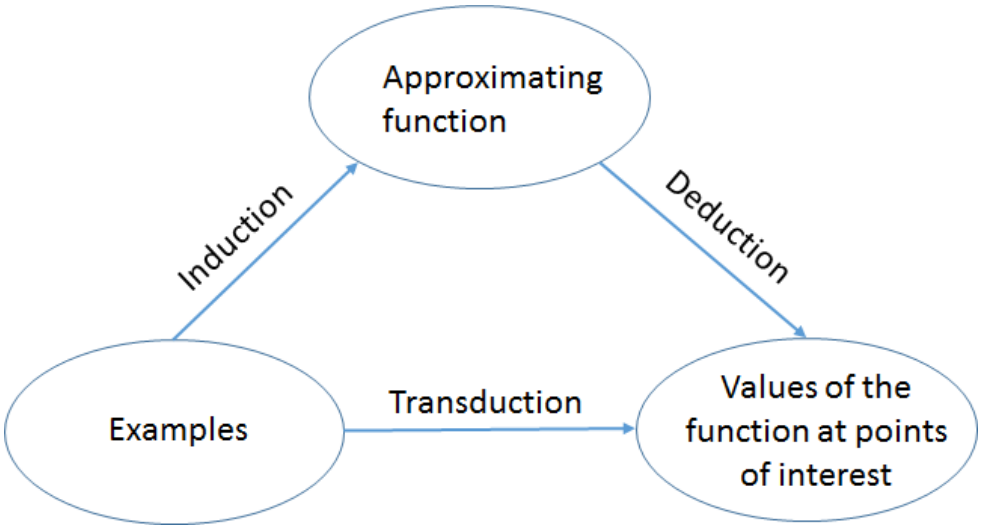

Figure 21 Transductive versus Inductive Inference

Consider the one dimensional observations shown in the Figure 22 Transductive Inference Example. The labeled values lie along the horizontal axis. If we were to use the labeled values to define the decision boundary, the dotted red line would be our decision boundary. If, on the other hand, we had access to a large set of unlabeled observations that conform to the distributions shown by the two Gaussian curves (to the left and right of the origin), we would choose the black line to be our decision boundary (based on the smoothness assumption).

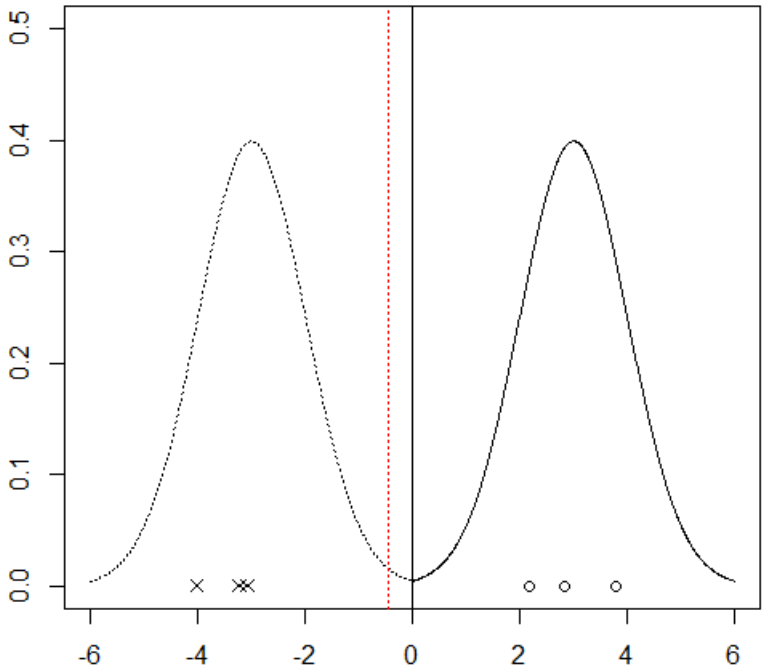

Figure 22 Transductive Inference Example

3.4 Strategies

The major learning strategies incorporated into this research include active learning and adversarial learning.

3.4.1 Active Learning

Active learning is the process of selecting which examples to use for training. Typically this is performed with "unlabeled" data. In this research, uncertainty sampling is used to select which observations to label for training. Clustering is performed using the Partitioning Around Medoids (PAM) algorithm on unlabeled data to identify cluster prototypes, to be labeled and used as training data for an initial classification model. This

initial classification model is then used to provide tentative labels for the remaining unlabeled training examples. The initial classification model is said to be most uncertain of a classification label for an unlabeled example if the estimated probability of class membership is the same for both classes; i.e. Probability(Positive Class | Observation) = Probability(Negative Class | Observation) = 0.5. Observations where the initial classification model is most uncertain about the label are selected for annotation, and all annotated data is then used to construct an updated classification model. Given a cluster prototype from (the center of) each of the two classes shown on the left and the right side of the Figure 23 Active Learning Example, uncertainty sampling will help to quickly refine the decision boundary after annotating only one new observation. For both the figure on the left and the figure on the right, the classifier will initially be most uncertain about observations along the vertical line x=0; however, after one of these observations is labeled, the classifier for the figure on the right will then be most uncertain about observations along the vertical line x= -1.5.

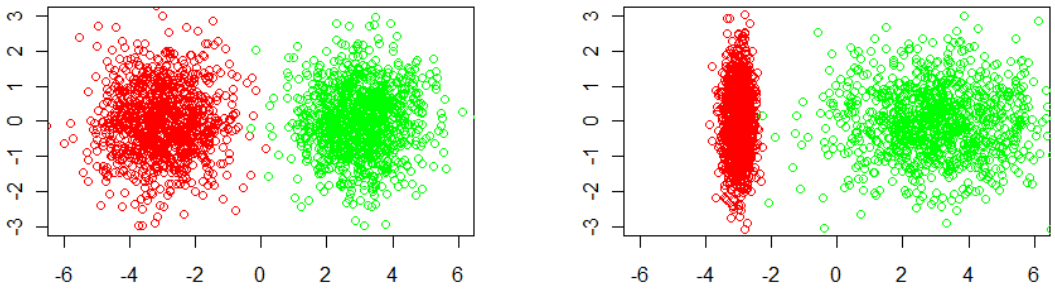

Figure 23 Active Learning Examples

3.4.2 Adversarial Learning

Adversarial learning can refer to an adversary's learning strategy. This is a reference to an adversary's use of machine learning to learn to avoid detection [40]. For example, for public e-mail service providers such as Hotmail, Yahoo! Mail, or Gmail, a spammer can open accounts and learn which message features separate messages identified as spam from messages not identified as spam. The spammer can then remove or obfuscate features that make a message more likely to be identified as spam (e.g. changing "free" to "f-r-e-e") or add features that make a message less likely to be identified as spam (e.g. text from a recent news article). For private e-mail service providers, such as a university, a spammer can include "beacons" (e.g. one pixel images) that let the spammer know which messages were viewed.

More commonly, however, adversarial learning refers to machine learning methods designed to negatively impact a spammer's ability to avoid detection. As in this research, the proposed machine learning methods are evaluated by modifying the test data in a manner designed to illustrate adversarial manipulation. This manipulation may include modifying the features to avoid detection, or modifying the annotations used to train detection systems. Both are addressed as part of this research.

4. PERFORMANCE EVALUATION

This chapter describes the variety of metrics that have been used to evaluate the proposed methods as well as the experimental design.

4.1 Metrics

For binary classification problems (such as spam detection, phishing detection, and fraud detection), there are several commonly used evaluation metrics which are based upon a 2 x 2 contingency table known as the Table 10 Confusion Matrix [41]. It is called a confusion matrix because it captures how often a binary classification model confuses members of one class for members of the other class.

Table 10 Confusion Matrix

		Predict ($\hat{y}$)	
		Class == Positive	Class == Negative
Actual (y)	Class == Positive	# True Positive (TP)	# False Negative (FN)
	Class == Negative	# False Positive (FP)	# True Negative (TN)

The confusion matrix captures counts for four possible situations when evaluating a "test" set:

1. True Positive (TP): a member of the positive class is predicted to be a member of the positive class
2. False Positive (FP): a member of the negative class is predicted to be a member of the positive class
3. False Negative (FN): a member of the positive class is predicted to be a member of the negative class
4. True Negative (TN): a member of the negative class is predicted to be a member of the negative class

There are several common metrics derived from the confusion matrix. These include Equation 4.1 Accuracy, Equation 4.2 Precision, Equation 4.3 Recall (aka Sensitivity or True Positive Rate), Equation 4.4 Specificity, Equation 4.5 False Positive Rate, and Equation 4.6 F Measure.

Equation 4.1 Accuracy

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} = Probability(\hat{y} == y)$$

Equation 4.2 Precision

$$Precision = \frac{TP}{TP + FP} = Probability(y == Positive \mid \hat{y} == Positive)$$

Equation 4.3 Recall (aka Sensitivity or True Positive Rate)

$$Recall = \frac{TP}{TP + FN} = Probability(\hat{y} == Positive \mid y == Positive)$$

Equation 4.4 Specificity

$$Specificity = \frac{TN}{TN + FP} = Probability(\hat{y} == Negative \mid y == Negative)$$

Equation 4.5 False Positive Rate

$$False\ Positive\ Rate = \frac{FP}{FP + TN} = Probability(\hat{y} == Positive \mid y == Negative)$$

Equation 4.6 F Measure

$$F = \left(\frac{Precision^{-1} + Recall^{-1}}{2}\right)^{-1} = \frac{Precision * Recall}{\left(\frac{Precision + Recall}{2}\right)} = \frac{TP + TP}{TP + FP + TP + FN}$$

As shown in Equation 4.6 F Measure, the F measure is a function of both precision and recall. It is simply the harmonic mean of precision and recall, which can also be viewed as the ratio of the product of precision and recall (the squared geometric mean) to the arithmetic mean of precision and recall. The F measure is bounded as shown in the Equation 4.7 Bounds for the F Measure.

Equation 4.7 Bounds for the F Measure

$$min(Precision, Recall) \leq F\ Measure \leq mean(Precision, Recall)$$

Another important evaluation tool is the Receiver Operating Characteristic (ROC) curve. For any classifier that generates a score (e.g. a probability or a signed distance from the decision boundary), an ROC curve can be generated by plotting the false positive rate on the horizontal axis and the true positive rate on the vertical axis while moving the classification threshold from positive infinity to negative infinity. For

example, consider the example ROC curves in the Figure 24 Example Receiver Operating Characteristic (ROC) Curves. The "Perfect" classifier (shown by the dotted line) is able to achieve a true positive rate of 100% with a false positive rate of 0%, while the true positive rate is always equal to the false positive rate for the "Random" classifier (shown by the dashed line). The Area Under the Curve (AUC) for the ROC curve is the probability that a randomly selected positive class member is assigned a higher positive class score than a randomly selected negative class member. The value of the AUC ranges from 0 (indicating the classifier has a false positive rate of 100% with a true positive rate of 0%) and 1 (indicating the classifier has a false positive rate of 0% with a true positive rate of 100%). The AUC is simply a normalized version of the "U" statistic from the two sample non-parametric Mann-Whitney test [42] of the research hypothesis that positive class members are assigned higher positive class scores than negative class members (normalized by the product of the count of positive class examples and the count of negative class examples: the number of possible pairs).

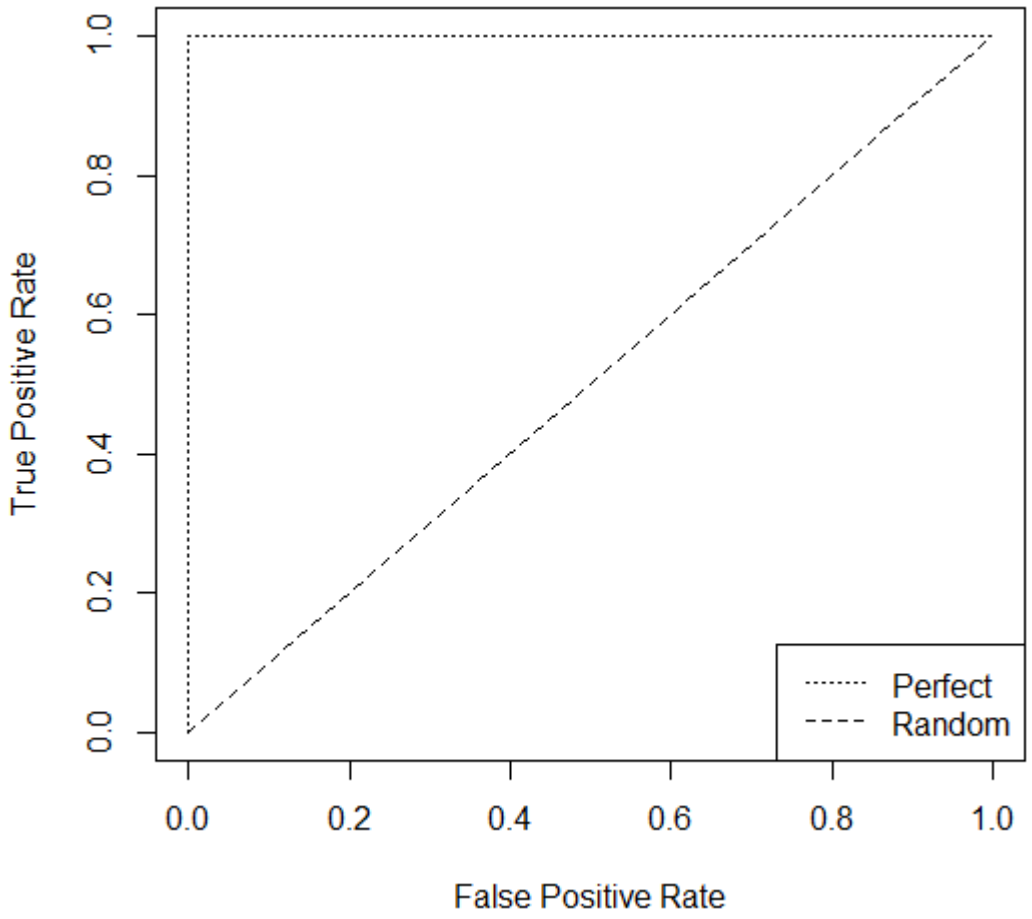

Figure 24 Example Receiver Operating Characteristic (ROC) Curves

4.2 Experimental Design

While learning parameter optimization is performed using cross validation of the training data, performance assessment is accomplished using sequestered test data. In a data streaming environment, where observations are time-ordered, the data is partitioned so that the training data precedes the testing data (so that emerging patterns of behavior cannot be observed as part of the training set).

5. SPAM DETECTION

Spam detection is an active area of research, as there are publicly available corpora and spammers have financial incentive to engage in adversarial behavior. Section 5.1 describes the use of clustering and active learning to reduce the amount of annotated training data. Section 5.2 describes the use of Social Network Analysis (SNA) features to improve spam detection. Section 5.3 describes the use of random subspace projections to construct an efficient spam detector that is robust to small changes in the underlying data distribution. Section 5.4 describes the use of repeated estimations of the decision boundary to identify mislabeled training examples.

5.1 Clustering and Active Learning

This research is focused on efficient construction of effective models for spam detection. Clustering messages allows for efficient labeling of a representative sample of messages for learning a spam detection model using a Random Forest for classification, and active learning allows for quickly refining the classification model. Results are illustrated for the 2007 TREC Public Spam Corpus. The area under the Receiver Operating Characteristic (ROC) curve is competitive with other state-of-the-art solutions while requiring much fewer labeled training examples.

Undesired, unsolicited e-mail is a nuisance for its recipients; however, it also often presents a security threat. For example, it may contain a link to a phony website intending to capture the user's login credentials (identity theft, phishing), or a link to a website that installs malicious software (malware) on the user's computer. Installed malware can be used to capture user information, send spam, host malware, host phish, or conduct denial of service attacks as part of a "bot" net. While prevention of spam transmission would be ideal, detection allows users and e-mail providers to address the problem today.

5.1.1 Background

Traditional machine learning techniques involve having a user label examples of both spam and ham (not spam) messages so that a computer algorithm can learn to identify unwanted e-mail [43] [44] [45]. For e-mail systems that process large quantities of messages, it is practically impossible to label all messages processed for some time period. In order to reduce the burden, the messages to be labeled are often selected randomly (passive learning), or they are selected by computer algorithm (active learning) [46]. Two common criteria used to select examples for training a pattern recognition model include:

- density based selection: a small set of representative examples are chosen for labeling, based on how well the examples characterize the data

- uncertainty based selection: an initial machine learning model is constructed from a labeled sample of examples, then the algorithm asks for labels for unlabeled examples where it is most uncertain about the class label

5.1.2 Proposed Method

The proposed method relies on five basic steps:

1. computing term frequency and inverse document frequency representation for messages
2. clustering a sample of messages from the training pool and obtaining labels for cluster medoids
3. constructing an initial Random Forest for spam detection
4. requesting labels for additional training examples based on uncertainty
5. refining the Random Forest model

The Figure 25 Clustering and Active Learning Training Process illustrates this work flow.

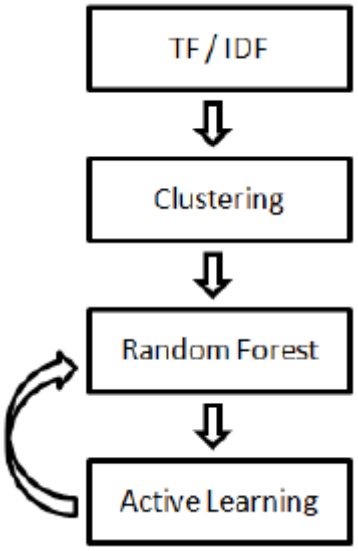

Figure 25 Clustering and Active Learning Training Process

RFC 822 [13] (Internet) e-mail messages are divided into two sections. The lines above the first blank line are headers, while the lines below the first blank line comprise the message body. As shown in the Figure 26 Example of RFC822 Message Headers, common headers include the "From", "To", "Subject", "Date", "Return-Path", and "Received" lines. Other headers include the "Content-Type" and "X-Mailer" lines.

```
From RickyAmes@aol.com  Sun Apr  8 13:07:32 2007
Return-Path: <RickyAmes@aol.com>
Received: from 129.97.78.23 ([211.202.101.74])
      by speedy.uwaterloo.ca (8.12.8/8.12.5)
      with SMTP id l38H7G0I003017;
      Sun, 8 Apr 2007 13:07:21 -0400
Received: from 0.144.152.6 by 211.202.101.74;
      Sun, 08 Apr 2007 19:04:48 +0100
Message-ID: <WYADCKPDFWWTWTXNFVUE@yahoo.com>
From: "Tomas Jacobs" <RickyAmes@aol.com>
Reply-To: "Tomas Jacobs" <RickyAmes@aol.com>
To: the00@speedy.uwaterloo.ca
Subject: Generic Cialis, branded quality@
Date: Sun, 08 Apr 2007 21:00:48 +0300
X-Mailer: Microsoft Outlook Express 6.00.2600.0000
MIME-Version: 1.0
Content-Type: multipart/alternative;
      boundary="--8896484051606557286"
X-Priority: 3
X-MSMail-Priority: Normal
Status: RO
Content-Length: 988
Lines: 24
```

Figure 26 Example of RFC822 Message Headers

Each message was converted to a set of normalized term frequency, inverse document frequency (TF/IDF) features [47]. Tokens were extracted from messages by converting the characters to lower case, separating on symbolic ASCII characters (such as the period and the exclamation point) and white space, and prefixing each token from a header with the name of the header. For example, the tokens for the headers in the Figure 26 Example of RFC822 Message Headers would include "subject:cialis" and "x-mailer:outlook". In order to reduce noise, only tokens that occur in at least 1% of the messages in the training pool and at most 99% of the messages in the training pool were retained.

Clustering has been combined with active learning in other application domains [48]. In this work, clustering is used to select an initial set of e-mail messages to be

labeled as training examples. The Partitioning Around Medoids (PAM) algorithm [49] was used to cluster a uniform random sample of 25% of the messages in the training pool. PAM is an implementation of the k-medoids algorithm. PAM selects the "k" most centrally located messages for its initial model, then iteratively assigns other messages to the nearest medoid and updates the medoid for each cluster. It is similar to the k-means clustering algorithm, but is less sensitive to the presence of outliers (unusual messages). The parameters for the model include the choice of distance measure and the number of clusters, "k". For these experiments, Euclidean distance was used to measure the difference between e-mail messages, and "k" was chosen to be the number of messages to be labeled. For example, if we wanted to label only 10 messages for our initial spam detection model, "k" was chosen to be 10.

After the cluster prototype messages were selected for training, feature selection and model selection were performed using leave-one-out cross validation. In repeated experiments, the model with the best performance was a Random Forest [50]. Table 11 Comparison of Cross Validation Performance shows a comparison of cross validation performance using 10 cluster prototypes for training. The performance measure is the Area Under the receiver operating characteristic Curve (AUC) [51].

Table 11 Comparison of Cross Validation Performance

Method	AUC
Random Forest	95.2%
Naïve Bayes	66.7%
SVM	66.7%
kNN	66.7%

A Random Forest is an ensemble of decision trees that vote on the spam/ham label for new messages. For each tree, a bootstrap sample is drawn from the labeled data and a decision tree is constructed by considering a random subset of features for each decision node in the tree. An example of a decision would be: "normalized TF/IDF value for token" > threshold. The strengths of the Random Forest method include feature selection and consideration of many feature subsets (instead of focusing on just a few features that best separate the training data). The key parameters for the Random Forest model include the number of trees to build and the number of features to consider for each decision node: 1,000 trees and 13 features were used as parameter values for these experiments.

Once the initial Random Forest model is constructed, additional messages are selected for labeling by choosing examples from the training pool where the probability of spam assigned by the Random Forest model is closest to 0.5. The probability of spam is computed as the proportion of decision trees assigning the spam label. In some sense, this is a combination of an uncertainty sampling [52] strategy (selecting the example the

classifier is most uncertain about) and a query by committee [53] strategy (where the uncertainty is correlated to the variance of an ensemble of decision trees). Once labeled, the selected messages are then added to the cluster prototypes and the Random Forest is retrained.

5.1.3 Experimental Design

The 2007 TREC Spam E-mail Corpus [54] contains 75,419 messages received by an e-mail server at the University of Waterloo between April 8 and July 6, 2007. For the experimental results reported here, the first week of data (9,535 messages) was used as a training pool and the next 12 weeks of messages were used for testing. Of the 573,670 distinct tokens found in the training pool, 4,515 were retained after removing tokens that appear in too few or too many messages; i.e. the retained tokens had to appear in at least 1% of the messages (to avoid over fitting) and at most 99% of the messages (to avoid "stop" words).

5.1.4 Performance Evaluation

An example of a cluster prototype for a large, well separated, compact cluster associated with a six hour phishing campaign is shown in the Figure 27 Cluster Prototype for a Phishing Campaign.

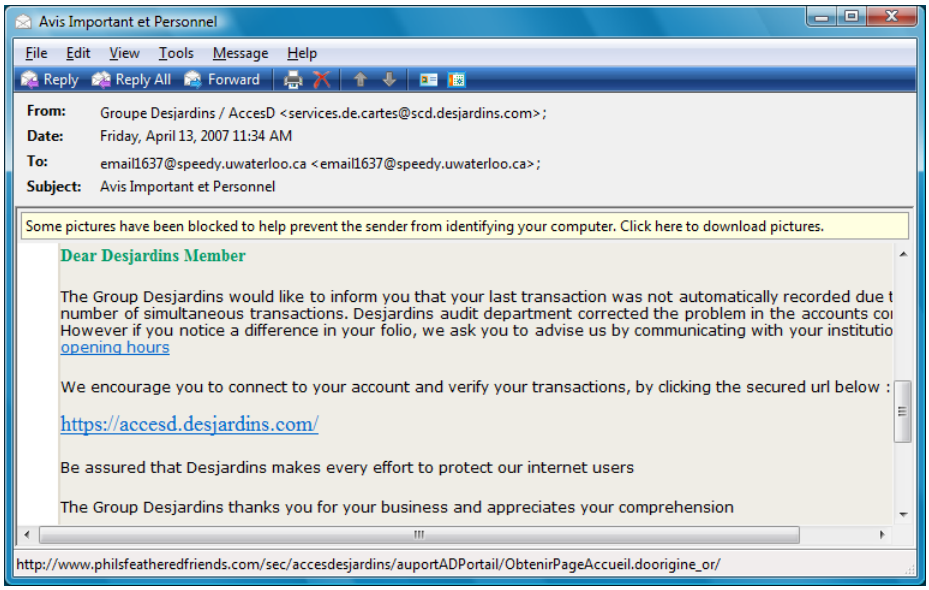

Figure 27 Cluster Prototype for a Phishing Campaign

The Receiver Operating Characteristic curves in the Figure 28 ROC Curves: Cluster 10 versus Random 10, Figure 29 ROC Curves: Cluster10 + Uncertain 10 versus Random 10 + Uncertain 10, and Figure 30 ROC Curves: Cluster 100 versus Random 100 graphs compare the performance of the following message selection algorithms on the test set:

- Cluster10 (dotted): unlabeled messages in the training pool were clustered into 10 groups and the medoids were selected for annotation [Area Under the Curve (AUC): 91.3%]
- Random10 (solid): 10 messages from the training pool were randomly selected for annotation [AUC: 68.2%]

- Cluster10 + Uncertainty10 (dotted): Cluster10 was used to construct an initial model, then 10 more messages from the training pool were selected based on the assigned probability of spam [AUC: 92.9%]
- Random10 + Uncertain10 (solid): Random10 was used to construct an initial model, then 10 more messages from the training pool were selected based on the assigned probability of spam [AUC: 65.7%]
- Cluster100 (dotted): unlabeled messages in the training pool were clustered into 100 groups and the medoids were selected for annotation [AUC: 99.7%]
- Random100 (solid): 100 messages from the training pool were randomly selected for annotation [AUC: 98.6%]

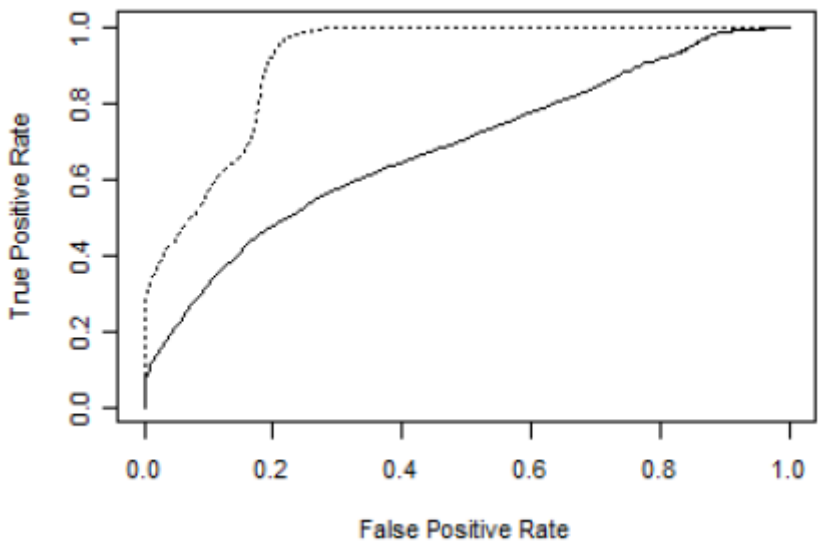

Figure 28 ROC Curves: Cluster 10 versus Random 10

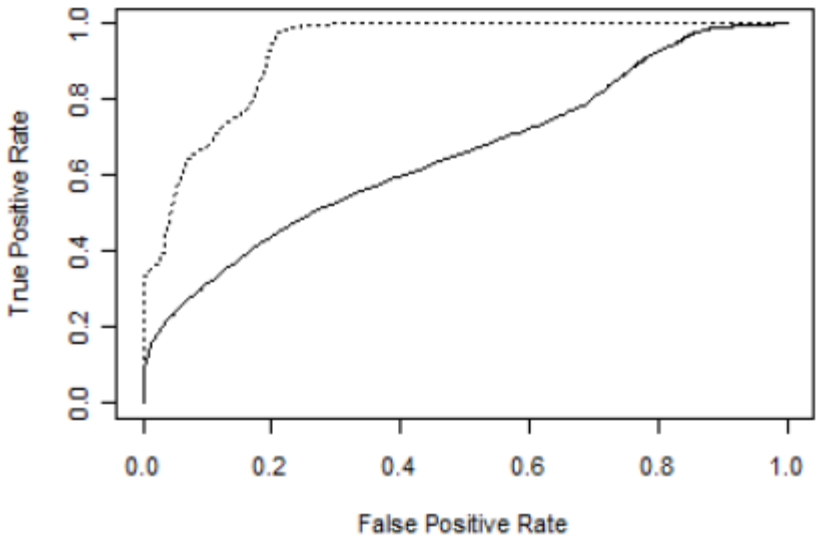

Figure 29 ROC Curves: Cluster10 + Uncertain 10 versus Random 10 + Uncertain 10

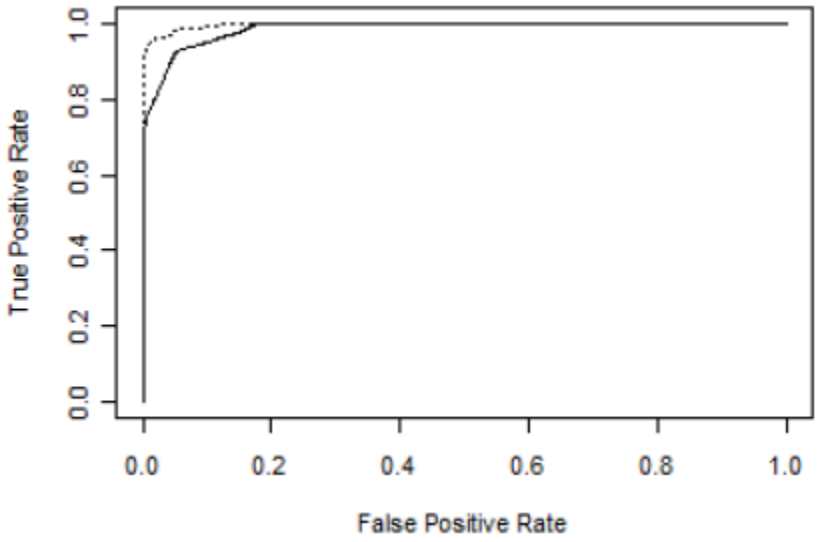

Figure 30 ROC Curves: Cluster 100 versus Random 100

The most novel observation from this work is the ability of cluster prototypes to efficiently represent compact, well separated clusters. For example, a burst of 2,532 messages from April 13th (over 25% of the training pool) can be effectively represented

by a single prototype (file=inmail.5413). This cluster was a phishing attempt disguised as an important notice. Another example of a compact, yet well separated, cluster was an advertisement for a pharmaceutical product (represented by file=inmail.3258).

Cluster prototypes provide a useful representation of the training pool for the Random Forest algorithm. Choosing an initial set of messages for labeling based on cluster prototypes, then choosing an additional set of messages for model refinement based on uncertainty offers improved performance. With as few as 100 labeled messages from one week of data, performance is competitive with state-of-the-art results [45].

Possible extensions of this work include selecting messages for labeling based on the emergence of new clusters in the data stream for a deployed model. The representation could be enhanced to include information about links and images contained in the messages.

5.2 Social Network Analysis Features

Content filtering is a popular approach to spam detection. It focuses on analysis of the message content to identify spam. In this research, we evaluate the use of social network analysis measures to improve the performance of a content filtering model. By measuring the degree centrality of message transfer agents, we observed performance improvements for spam detection in repeated experiments; e.g. a 70% increase in the proportion of spam detected with a false positive rate of 0.1%. We were also able to use anomaly detection to identify mislabeled messages in a publicly available spam data set.

Messages with unusual path lengths between the sender's message transfer agent and the recipient's message transfer agent turned out to be spam.

Unwanted e-mail, also known as spam, affects all e-mail users on a routine basis. Some spam messages get delivered to the user's inbox. The content of these spam messages includes advertising for pharmaceuticals, jewelry, electronics, software, loans, stocks, gambling, weight loss, and pornography; as well as malware and phishing (identify theft) lures. Wanted e-mail, also known as ham, may also be incorrectly junked as spam. And all users experience delays in message delivery as the e-mail system works to determine which messages are spam and which messages are ham.

Content filters parse incoming messages looking for the presence of particular features in the message content to determine whether a message is ham or spam. For example the term "alert" may occur more frequently in ham (news alerts), while the term "http" may occur more frequently in spam (links). There are, of course, no magic terms that allow a content filtering system to perfectly separate spam and ham. Even if there were, spammers would quickly find these terms and adapt; i.e. start using the ham terms and stop using (or obfuscate) the spam terms.

5.2.1 Background

Social Network Analysis focuses on measuring various aspects of entities and the relationships between them. This includes identifying "central" nodes within a network [55] and determining the distance between nodes [56]. The entities of interest are often

people and the relationships of interest are often social interactions, but the concepts are easily transferred to computers and the e-mail connections between them.

Many spam filtering systems augment their content filtering system with a block list, for known spam senders. Of course, the spammers have adapted to this system as well. By using malware to infect computer systems, spammers can then use the infected computers to send spam for them. These infected systems are called proxies, because they send the spam on behalf of the spammer. In order to hide the source of these messages, the proxies often route the spam messages through "open relays" [57]. An "open relay" is a Message Transfer Agent (Simple Mail Transfer Protocol [12] server) that will allow any Internet user to send e-mail through it. The Figure 31 Simple Network Connections Between a Sender and Receiver show a typical configuration between sender and recipient. For example, when user@example.edu sends an e-mail to ddebarr@gmu.edu, user sends the message to mta.example.edu and mta.example.edu forwards the message to mta.gmu.edu. The focus of this work is to evaluate whether the degree centrality of the sender MTA and the path length are useful for helping to distinguish spam from ham, assuming that "open relays" are likely to have many more senders than other sender MTAs.

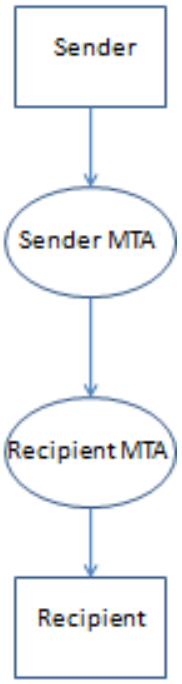

Figure 31 Simple Network Connections Between a Sender and Receiver

The connections between computers are recorded in message "headers", a portion of the message typically not seen by the sender or the recipient. The format of these headers vary from host to host, but they typically have the following form:

Received: from sender_mta_name ([sender_mta_address]) by recipient_mta_name

The sender_mta_address is recorded as an Internet Protocol (IP) address; e.g. "10.1.1.101", the network address of the sending computer. A typical e-mail address will contain two "received" headers. The first recorded by the sender MTA, and the second recorded by the recipient MTA.

5.2.2 Proposed Method

Our proposed spam detection filter is trained with both content filtering features and our proposed social network analysis features. The content of the messages is divided into tokens, and the term frequency and inverse document frequency of the tokens are used to train the spam detection filter. For example, if the term "alert" is found 3 times in the content of a particular message and it occurs in 100 out of 10000 messages in the training corpus, this token is represented by the number 13.8 [3 * log(10000/100)]. The token values for each message are then divided by the square root of the sum of squared token values, in order to mitigate the effect of differences in message length.

The proposed social network analysis features include the degree centrality of the sender's MTA as well as the path length between sender and recipient. To help identify possible open relays, we used the degree centrality of the sender's MTA. This was computed as the indegree of the sender MTA (known as "prestige" in social network analysis). For example, the following "received" headers form a link between "192.168.1.201" and "10.1.1.101":

Received: from sender ([192.168.1.201]) by sender_mta

Received: from sender_mta ([10.1.1.101]) by recipient_mta

To assess whether the sender_mta of an incoming message is an open relay, we used the "number of distinct sender subnets for the sender_mta during the previous week of observed e-mail traffic" (indegree) as a feature for training the spam detection filter.

The first three components (bytes) of the IP address were used as a subnet identifier. For example, if sender_mta received e-mail from 1.2.3.4, 1.2.3.5, and 6.7.8.9, the indegree would be recorded as 2 (for subnets 1.2.3.* and 6.7.8.*). We also used the number of "received" headers as a training feature, assuming that unusual path lengths from sender to recipient are more likely to be spam-related.

Machine learning algorithms are often used to train a computer to recognize patterns within data. For spam filtering, we provide the computer with examples of both spam and ham so that it can learn to distinguish between the two message categories. The machine learning algorithm known as Logit Boost is among the best in terms of constructing an effective spam detection filter. A trained model consists of a set of conditions and weights that are used to evaluate each message. For example, a condition might be "indegree for sender_mta > [learned threshold]" and the associated weight might be some positive value. Because Logit Boost is an additive logistic regression algorithm, these weights are used directly to estimate the probability that a message is spam:

$$Probability(Spam) = \frac{1}{1 + exp\left(-\sum_{i=1}^{p} weight_i * indicator_i\right)}$$

If the sum of weights is 0, then the Probability(Spam) is 50%; if the sum of weights is negative, then the Probability(Spam) < 50%; and if the sum of weights is positive, then the Probability(Spam) > 50%.

5.2.3 Experimental Design

The publicly available corpus from the "Spam Track" of the 2007 Text Retrieval Conference (TREC) was used for our experiments. It consists of 75,419 messages received by two e-mail servers at the University of Waterloo, from April 8 to July 6 2007. Messages received from Monday April 16 to Sunday April 22 were used for training, while messages received from April 23 to July 6 were used for testing. Table 12 Message Counts for SNA Feature Evaluation shows the number of messages used for training and testing.

Table 12 Message Counts for SNA Feature Evaluation

	Spam	Ham
Train	5,162	2,171
Test	37,116	21,120

Because obtaining ham/spam labels for messages requires manual intervention, only a 1% sample of the training set was actually used for constructing a Logit Boost model. The training set was randomly divided into 5 subsets, then each subset of messages was clustered based on TF/IDF (term frequency, inverse document frequency) representation. The Partitioning Around Medoids (PAM) was used for clustering, and a total of 73 cluster prototypes from each of the 5 subsets were used to train Logit Boost models. The number of clusters was chosen based on the average silhouette value. Cross validation was used to optimize the parameters of the Logit Boost learning algorithm as

well. The scaling coefficient was set to 0.5 and the number of iterations was set to 40 for each model. Each trained model was then evaluated using the entire test set.

5.2.4 Performance Evaluation

Performance was evaluated using the area under the Receiver Operating Characteristic (ROC) curve (AUC). The ROC curve is generated by plotting the False Positive (FP) rate on the horizontal axis and the True Positive (TP) rate on the vertical axis while changing the spam detection threshold from probability=1 to probability=0. The Table 13 AUC Comparisons for SNA Feature Evaluation compares the performance of the models with and without Social Network Analysis (SNA) features for each of the 5 training subsets. Each row compares the performance of the "Content Features Only" approach to the "Content + SNA Features" approach, using precisely the same training messages from each subset.

Table 13 AUC Comparisons for SNA Feature Evaluation

	Content Features Only	Content + SNA Features
Trial 1	93.2%	94.9%
Trial 2	94.3%	97.1%
Trial 3	97.2%	98.4%
Trial 4	95.9%	96.6%
Trial 5	95.0%	95.5%

The Figure 32 ROC Curves: With versus Without SNA Features show the difference in the ROC curves for subset 3. The solid line shows the performance of the "Content Features Only" model, while the dotted line shows the performance of the "Content + SNA Features" model. The Figure 33 Region of Interest for ROC Curves: With versus Without SNA Features zooms in on the upper left hand corner of the Figure 32 ROC Curves: With versus Without SNA Features, highlighting the difference in performance when low false positive rates are required. The Table 14 Comparison of Spam Detection Rates: With versus Without SNA Features compares the percentage of spam detected for selected false positive thresholds. For the lower false positive rates, the difference in performance is very significant. For example, with a false positive rate of 1 in 1000 ham messages (incorrectly identified as spam), the "Content + SNA Features" model detects 1.7 times as many spam messages.

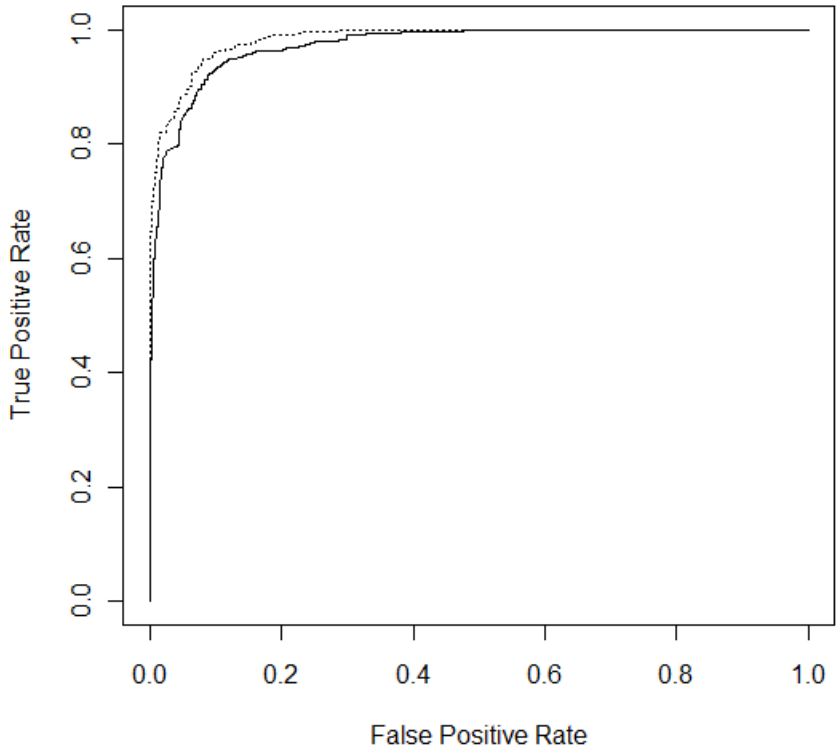

Figure 32 ROC Curves: With versus Without SNA Features

The "Content + SNA Features" model shown in the Figure 33 Region of Interest for ROC Curves: With versus Without SNA Features assigns weight +2.0 to the log odds of spam if the indegree of the sender's MTA was at least 15 (i.e. the sender's MTA was used by 15 or more subnets within the previous week). The weight -1.55 was assigned otherwise.

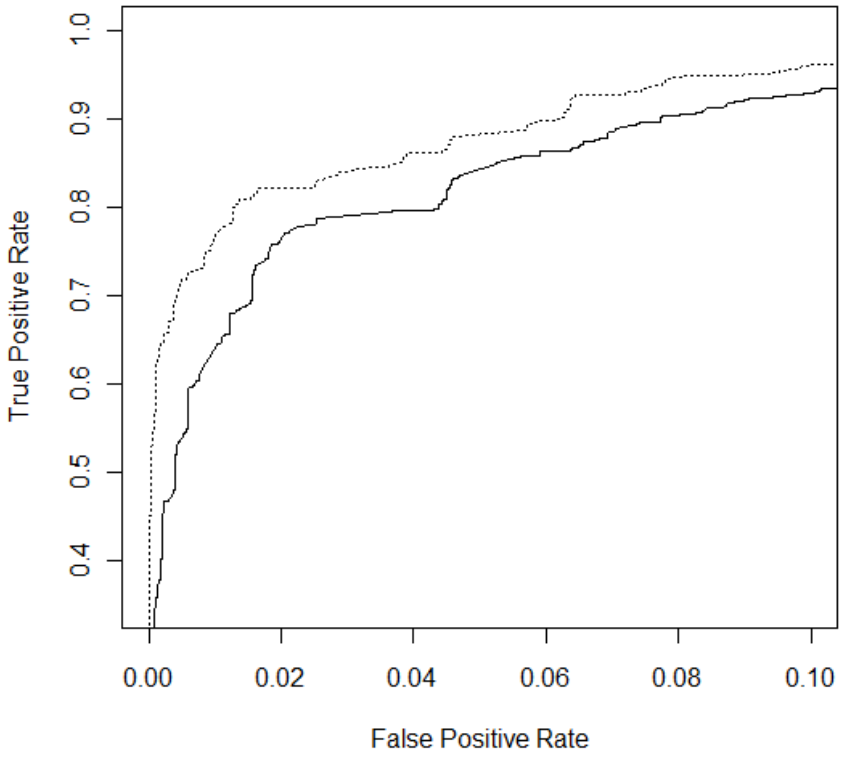

Figure 33 Region of Interest for ROC Curves: With versus Without SNA Features

Table 14 Comparison of Spam Detection Rates: With versus Without SNA Features

False Positive Rate	Content Features Only	Content + SNA Features
0.1%	35.73%	60.64%
0.2%	44.37%	64.56%
0.3%	46.66%	66.96%
0.4%	51.52%	69.07%
0.5%	53.78%	71.79%
1.0%	62.26%	76.78%
2.0%	76.05%	82.03%
3.0%	78.72%	83.90%
4.0%	79.55%	86.12%
5.0%	83.57%	88.00%

While the path length (number of "received" headers) was not emphasized in the Logit Boost models, it was useful for identifying test set messages that were incorrectly labeled. For example, there were 14 messages that had 12 or more "received" headers; but only 2 of them were labeled as spam. A review of the messages labeled as "ham" shows that these messages have been mislabeled: 36527, 37057, 39208, 39846, 39904, 40652, 41486, 43113, 44498, 45275, 46296, and 47221.

Figure 34 Spam Message Mislabeled as Not Spam shows the content of message 43113. It is an advertisement for Cialis and Viagra that has been sent to an e-mail list. This emphasizes the difficulty of obtaining accurate annotations for messages.

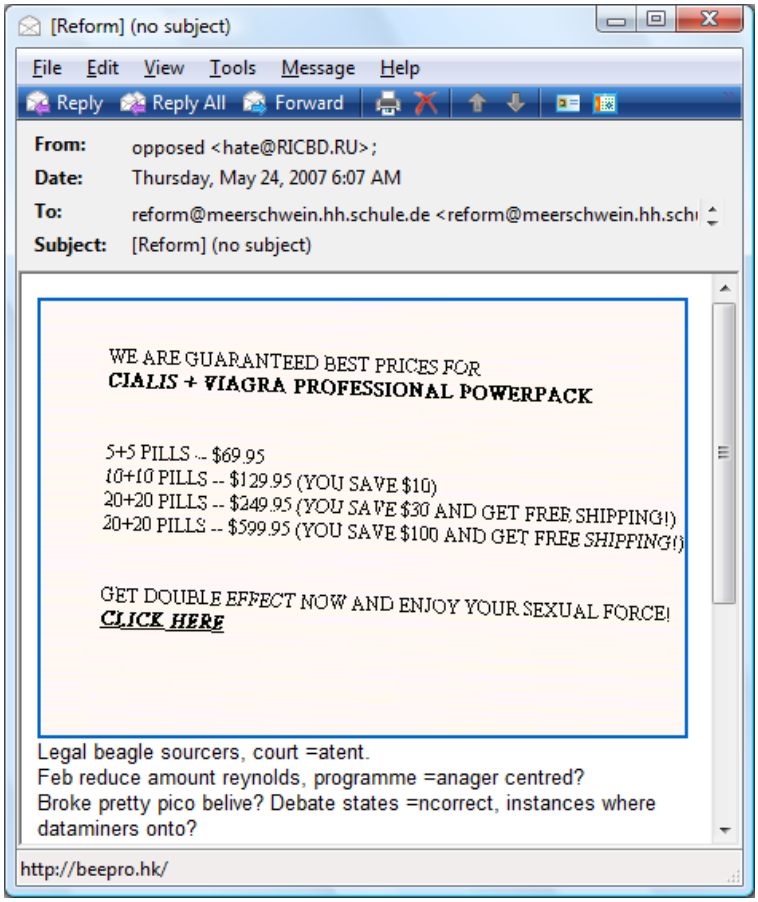

Figure 34 Spam Message Mislabeled as Not Spam

The use of social network analysis features significantly improved the performance of a content filter for spam detection. The indegree feature appears to be useful for identifying open relays, mail servers that will accept e-mail traffic from anyone (allowing spammers to hide their identity). It's possible the open relays that predominantly send spam could be added to a network address block list, thus avoiding the overhead associated with content filtering. Future work includes evaluating the use of

more complicated social network analysis features to characterize the network connectivity of mail transfer agents along the path between sender and recipient.

5.3 Random Boost

This research proposes two alternative methods of random projections and compares their performance for robust and efficient spam detection when trained using a small number of examples. Robustness refers to learning and adaptation leading to a high level of performance despite data variability, while efficiency is concerned with (i) the complexity of the detection method employed; and (ii) the amount of training resources used for training and retraining. The first method, Random Project, employs a random projection matrix to produce linear combinations of input features, while the second method, Random Boost, employs random feature selection to enhance the performance of the Logit Boost algorithm. Random Boost is, in fact, a combination of Logit Boost and Random Forest. Experimental results, using TREC and CEAS as challenging spam benchmark sets, show that the Random Boost method significantly improves the performance of the spam filter compared to the Logit Boost algorithm (e.g., a 5% increase in AUC, which is the area under the Receiver Operating Characteristic curve), and yields similar classification accuracy compared to the Random Forest method but using only one-fourth the runtime complexity of the Random Forest algorithm. Additionally, the Random Boost algorithm also reduces training time by two orders of magnitude compared to Logit Boost, which becomes important during retraining on the

ever-changing data streams, including adapting to adversarial tactics and "noise" injected by spammers.

Computer-based spam detection methods are required to handle the onslaught of electronic junk mail. For a spam filter to be effective, however, it needs to be customized for the environment in which it will operate. This means one or more people need to provide "labeled" training messages, including examples of both "spam" and "not-spam" messages. It can take much time to label a substantial corpus of training messages. For this reason, it is preferable to create the initial spam detection model with as few labeled training messages as possible. Unfortunately, the use of fewer training messages causes a problem for commonly used greedy optimization algorithms, such as logistic regression models and decision trees, because the greedy optimization algorithms focus on using a few (best) features for detecting spam and ignore other possible indicators.

The work described here proposes and evaluates two alternatives forms of random projection, to generate robust spam filters that can be trained from a small number of examples. The Random Project method uses linear combinations of the input features, while the Random Boost method uses random feature selection. This makes the Logit Boost learning method, which underlies Random Boost, more robust against noise by properly choosing the features used for distinguishing spam from not spam messages. The terms spam detection and spam filters are used interchangeably.

5.3.1 Background

Many machine learning methods have been applied to the problem of spam detection including contents filtering. The typical approach to contents filtering is to break the text of the message into words, and then use frequency information about the words as features for classifying the message. For example, terms such as "remove" and "free" may appear more often in spam messages, while terms such as "meeting" and "project" may appear more often in not-spam messages. One of the earliest known approaches involved the use of Naïve Bayes [58]:

Equation 5.1 Naive Bayes Estimation of the Probability of Spam

$$Prob(Spam|\boldsymbol{x}_i)$$

$$= \frac{Prob(Spam)\prod_{f=1}^{d} Prob\left(x_{i,f}\middle|Spam\right)}{Prob(Spam)\prod_{f=1}^{d} Prob\left(x_{i,f}\middle|Spam\right) + Prob(Not\ Spam)\prod_{f=1}^{d} Prob\left(x_{i,f}\middle|Not\ Spam\right)}$$

where $x_{i,f}$ is feature f of vector x_i and d is the number of features in message $\boldsymbol{x}_i$.

While Naïve Bayes is computationally inexpensive to learn, it is very sensitive to the features used to construct the model. For example, use of correlated features degrades the performance of the model. Logistic regression and support vector machines have been shown to have better performance than Naïve Bayes [7]; however, their performance is also sensitive to the features used for constructing the spam detection model. A logistic regression model is, in fact, structurally equivalent to a Support Vector Machine (SVM) that uses a linear kernel. Given a vector $\boldsymbol{x}_i$, the structural form of a logistic regression model is based on the logistic function:

$$Prob(Spam|\boldsymbol{x}_i) = \frac{1}{1 + exp\left(-\sum_{f=1}^{d} weight_f feature_{i,f}\right)}$$

where $weight_f$ is a weight for feature $feature_{i,f}$ and there are d features in the logistic regression model. Positive weights increase the log odds of spam, while negative weights decrease the log odds of spam.

For a support vector machine with a linear kernel, the classification function is based on the sum of dot products between the vector to be classified and the support vectors that define the decision boundary:

Equation 5.2 Equivalence of Linear SVM and Logistic Regression Decision Functions

$$Sign\left(\sum_{i=1}^{v}\left(y_i\alpha_i\sum_{f=1}^{d}x_{i,f}s_{i,f}\right)\right) = Sign\left(\sum_{f=1}^{d}\left(x_{i,f}\sum_{i=1}^{v}y_i a_i s_{i,f}\right)\right)$$

where $s_{i,f}$ is feature f of support vector i, y_i is the class of the support vector (either +1 or -1), and α_i is a measure of how important support vector $\boldsymbol{s}_i$ is for defining the decision boundary. The $\sum_{i=1}^{v} y_i a_i s_{i,f}$ term in the SVM model is analogous to the $weight_f$ term in the logistic regression model.

Groups (ensembles) of classifiers have shown significant performance improvements over single models [59]. Two popular approaches to constructing ensembles of classification models are boosting and bootstrap aggregation (bagging). Boosting focuses on incrementally reducing bias, by reweighting training examples for the next ensemble model based on a measure of error for the current ensemble of models; while bagging focuses on incrementally reducing variance, by training each member of the ensemble from an independent bootstrap sample of the training data.

The Logit Boost algorithm is an implementation of boosting where training examples are reweighted based on the logistic loss function. Minimizing the logistic loss function is equivalent to maximizing the likelihood of the data:

Equation 5.3 Likelihood of the Observed Data

$$\prod_{i=1}^{n} \frac{1}{1 + exp\big(-y_i \, prob(spam|\boldsymbol{x}_i)\big)}$$

In fact, Logit Boost is a form of additive logistic regression; i.e., a logistic regression ensemble model where each new component model is focused on reducing the error of the current ensemble of models. The residual (remaining error) for each observation is computed as

Equation 5.4 Residual Error for Additive Logistic Regression

$$residual_i = \frac{y_i{}^* - prob(spam|\boldsymbol{x}_i, \ \ current\ model)}{prob(spam|\boldsymbol{x}_i, \ \ current\ model)\big(1 - prob(spam|\boldsymbol{x}_i, \ \ current\ model)\big)}$$

The Logit Boost model enjoys the same run-time performance as a logistic regression model with the added benefit of automated feature selection. Predictions are produced as follows:

$$prob(spam|\boldsymbol{x}_i) = \frac{1}{1 + exp(-\sum_{m=1}^{M} predict(residual_i \mid \boldsymbol{x}_i))}$$

In practice, regression stumps are often used to implement the $predict(residual_i \mid \boldsymbol{x}_i)$ functions; i.e., each model predicts the expected value of the

residual based on the condition that minimizes variance of the predictions. Here's an example of a regression stump for a Logit Boost model:

if freq("remove" | x_i) > 0.01 then return 1.91 else return -0.41

Equation 5.5 Example of a Regression Stump

The Logit Boost model is a greedy learning algorithm. At each step, it selects the feature that reduces the residual error the most. By consistently focusing on only the best feature for reducing error, the Logit Boost model is susceptible to making mistakes based on small amounts of noise related to the few features that are used as part of the model.

The Random Forest algorithm is an implementation of bagging where each tree in an ensemble of decision trees is constructed from a bootstrap sample of messages from the training set. Each bootstrap sample of messages is obtained by repeated random sampling with replacement until the size of the bootstrap sample matches the size of the original training set. When constructing each decision tree, only a randomly selected subset of features is considered for constructing each decision node. Of the "k" randomly selected features to consider for constructing each decision node, the yes/no condition that best reduces the entropy of the data is selected for the next node in the tree.

The entropy is largest when the classifier is most uncertain about whether a message is spam. Here's an example of a simple decision tree for spam detection:

```
if freq("remove") > 0.01 then
    return "spam"
else
    if freq("free") > 0.10 then
        return "spam"
    else
        return "not spam"
```

Figure 35 Example of a Decision Tree

The Random Forest algorithm, however, unlike other decision tree learning algorithms, recursively splits the data set until it is not possible to reduce the entropy of each leaf node (either because all the training set messages for a leaf node have the same classification or it is not possible to effectively separate the spam messages from the not-spam messages). The entire ensemble of decision trees is then used to classify new messages using a voting method. Each tree in the ensemble assigns a class label for a new message, so the estimated probability of spam is simply the proportion of trees that labeled the new message as spam. The Random Forest algorithm is robust against noise, because it makes use of many features for spam classification; however, the use of additional features substantially increases the runtime complexity of deployed models. In fact, the runtime complexity of the Random Forest algorithm is often several times larger than the runtime complexity for the Logit Boost algorithm.

5.3.2 Proposed Method

A random projection is typically used to reduce the dimensionality of a high dimensional input space in order to make learning the prediction model more efficient [60]. This certainly applies to spam detection, where a set of training messages typically consists of thousands of possible features (words used for distinguishing spam from not-spam messages). For our purposes, however, we are also hoping for an added benefit: by using features composed of linear combinations of the words, our model will be more robust because it will leverage more than just the few best features for distinguishing spam from not-spam. The spam detection model will also handle adversarial behavior where the spammer injects additional features as noise.

Random Project consists of using a matrix of random numbers to project input vectors to a new feature space. The Gaussian distribution is often used to generate these random numbers. The columns of the transform matrix are then normalized so length=1 for each vector. The input vectors are projected by multiplying each row of the input matrix by a column in the transformation matrix. This is equivalent to performing feature selection based on linear combinations of input features [61].

The number of columns in the transformation matrix determines the dimensionality of the new feature space. According to the Johnson-Lindenstrauss Lemma [62], if there are "n" rows and "d" columns in the input feature matrix and "k" columns in the transformation matrix, where $k > \frac{4 \log n}{\varepsilon^2/2 - \varepsilon^3/3}$ with $\varepsilon \in (0,1)$, then

Equation 5.6 Distance Bounds for Random Projection

$$Prob\left[(1-\varepsilon)\frac{k}{d}dist(\boldsymbol{x}_i,\boldsymbol{x}_j)^2 \le dist(\boldsymbol{x'}_i,\boldsymbol{x'}_j)^2 \le (1+\varepsilon)\frac{k}{d}dist(\boldsymbol{x}_i,\boldsymbol{x}_j)^2\right]$$

$$\ge 1 - exp\left(\frac{k}{2}\left(1-(1-\varepsilon)+ln(1-\varepsilon)\right)\right)$$

$$- exp\left(\frac{k}{2}\left(1-(1+\varepsilon)+ln(1+\varepsilon)\right)\right)$$

This says, with high probability, the squared distance between a pair of transformed vectors will be close to the squared distance between the original input vectors, with the distance scaled by k/d. Larger values of "k" make this more likely to be true.

We used a Gaussian random number generator, with mean=0 and standard deviation=1, to populate our transformation matrices. To illustrate this transform, consider two dimensional input vectors, where the features of the input vectors capture the number of words often associated with "spam" (like "remove" or "free") and the number of words often associated with "not spam" (like "meeting" or "project"). A simple one column transformation matrix could then be generated by generating two random numbers and dividing each number by the square root of the sum of the squared random numbers. Matrix multiplication would then be used to transform the original two dimensional input vectors to the transformed one dimensional values.

In this example, we have reduced the 2 dimensional input vectors to 1 dimensional transformed vectors. The number of columns in the transform matrix dictates the number of dimensions for the transformed vectors. According to the

Johnson-Lindenstrauss Lemma, the number of dimensions is chosen to bound the distance error for the transformed vectors. In practice, this parameter is chosen empirically by cross validation during the training process.

While lower dimensionality makes it faster to learn the detection model, by using a linear combination of the original input features to create new feature vectors, the transformation also makes the detection model more robust to changes in the input space. For example, suppose a greedy learning algorithm trained on the original input vectors simply uses only one of the features to distinguish the positive class from the negative class. Noise added to this single feature could cause the model produced by the greedy learning algorithm to misclassify a new message. By using a linear combination to produce the transformed vectors, our classifier thus employs additional features to become robust to possible noise injected by the adversary.

For the Random Project algorithm, transformed feature vectors are used in conjunction with the Logit Boost algorithm for constructing a spam filter. For training, the input vectors are multiplied by the transformation matrix, and then the transformed vectors are used for training the Logit Boost algorithm. For testing, the input vector for a new message is multiplied by the same transformation matrix, and then the transformed vector is classified using the Logit Boost algorithm.

The *Random Boost* algorithm proposed here is designed to ensure that the learning algorithm will consider more than just an optimally small set of features for making classification decisions. The Random Boost algorithm is a combination of the Logit Boost and Random Forest learning algorithms. The Random Boost algorithm is

identical to the Logit Boost algorithm with one exception, only a randomly selected subset of features are considered for constructing each regression stump to be added to the ensemble. The Figure 36 Random Boost Learning Algorithm contains pseudo-code describing the Random Boost algorithm.

for i = 1 to N (where N is the number of messages in the training set)

$p_i = 1/2$

for m = 1 to M (where M is the number of stumps to construct)

for i = 1 to N

$r_i = (y_i - p_i)/\big(p_i(1-p_i)\big)$ (where y_i is 1 if message is spam, 0 otherwise)

bestVar = ∞

randomly select *k* of the *d* features for evaluation

for each feature *f* in the randomly selected subset

for each possible split point *p* for feature *f*

$v_p = variance(r_i|value \leq p) + variance(r_i|value > p)$

if v_p < bestVar then

record feature, split point, and average r_i for each group as $w_m(\,)$

bestVar = v_p

for i = 1 to N

$p_i = \frac{1}{1+exp\left(-\sum_{j=1}^{m} w_j(x_i)\right)}$

Figure 36 Random Boost Learning Algorithm

The use of random subsets of features for constructing regression stumps can be viewed as a form of random subspace projection, where only one of the elements of a column of the transformation matrix is set to 1, while the remaining values are set to 0. Like the random projection algorithm presented earlier, the expected value of the distance between a pair of transformed vectors is the distance between the original input vectors scaled by k/d.

By definition, the expected value of a random variable is equal to the sum of products of the value and the probability of the value:

Equation 5.7 Expected Value of a Random Variable

$$E[X] = \sum_{i} prob(\boldsymbol{x}_i)\, \boldsymbol{x}_i$$

This means that the expected squared distance between two observations from a randomly selected subspace is equal to the squared distance between the two observations in the original input space, scaled by k/d:

Equation 5.8 Expected Distance Between Observations Using Random Subspace Projection

$$dist(\boldsymbol{x'}_i, \boldsymbol{x'}_j)^2 = \sum_{f=1}^{d} prob(f \text{ is selected}) \left(\boldsymbol{x}_{i,f} - \boldsymbol{x}_{j,f}\right)^2 = \sum_{f=1}^{d} \frac{k}{d} \left(\boldsymbol{x}_{i,f} - \boldsymbol{x}_{j,f}\right)^2$$

$$= \frac{k}{d} \sum_{f=1}^{d} \left(\boldsymbol{x}_{i,f} - \boldsymbol{x}_{j,f}\right)^2 = \frac{k}{d} dist(\boldsymbol{x}_i, \boldsymbol{x}_j)^2$$

The resulting model from the Random Boost learning algorithm is simply an additive logistic regression model, where the stumps make use of more features than the standard Logit Boost learning approach.

5.3.3 Experimental Design

The TREC 2007 and CEAS 2008 data sets were used to compare the performance of the Logit Boost, Random Forest, Random Project, and Random Boost algorithms. Both the TREC 2007 and CEAS 2008 corpora are challenging data sets collected for spam detection competitions. The TREC 2007 corpus consists of 75,419 messages collected at the University of Waterloo from April 4 to July 6, 2007 (3 months), while the CEAS 2008 corpus consists of 137,705 messages collected anonymously from August 5 to August 8, 2008 (3 days). For both data sets, the first 10% of the data set was used for learning, while the remaining 90% of the data set was used for testing.

The steps for generating the spam detection model include: 1) Construct TF/IDF feature vectors; 2) Cluster a random sample of 2,500 messages; 3) Label the medoids spam or not spam; and 4) Construct the classifiers using the medoids for training. The first 10% of the messages was used for learning. The size of the data set was chosen to obtain a representative sample; i.e., a diverse set of messages that would be similar to future messages.

Term Frequency / Inverse Document Frequency (TF/IDF) features were extracted by splitting each message into tokens based on spaces, tabs, and symbols (such as punctuation). The TF/IDF values for each message were computed, and the TF/IDF

vectors were scaled to unit length to reduce the effect of message length on the classification process.

The Partitioning Around Medoids (PAM) algorithm was used for clustering a random sample of 2,500 messages from the 10% of the messages used for learning. In our earlier experiments [1], the use of cluster medoids for training was shown to provide better performance than simple random sampling. The medoid for a cluster is the message that has minimum average distance to other messages in the cluster. The size of this sample (2,500 messages) was chosen to make it easy to evaluate many different values for the number of clusters to be constructed. The average "silhouette" value was used to choose the best number of clusters. The Figure 37 Average Silhouette Values for the TREC Data and Figure 38 Average Silhouette Values for the CEAS Data graphs show the average silhouette value for each cluster count evaluated: 40 clusters appeared optimal for the TREC data, while 30 clusters appeared optimal for the CEAS data.

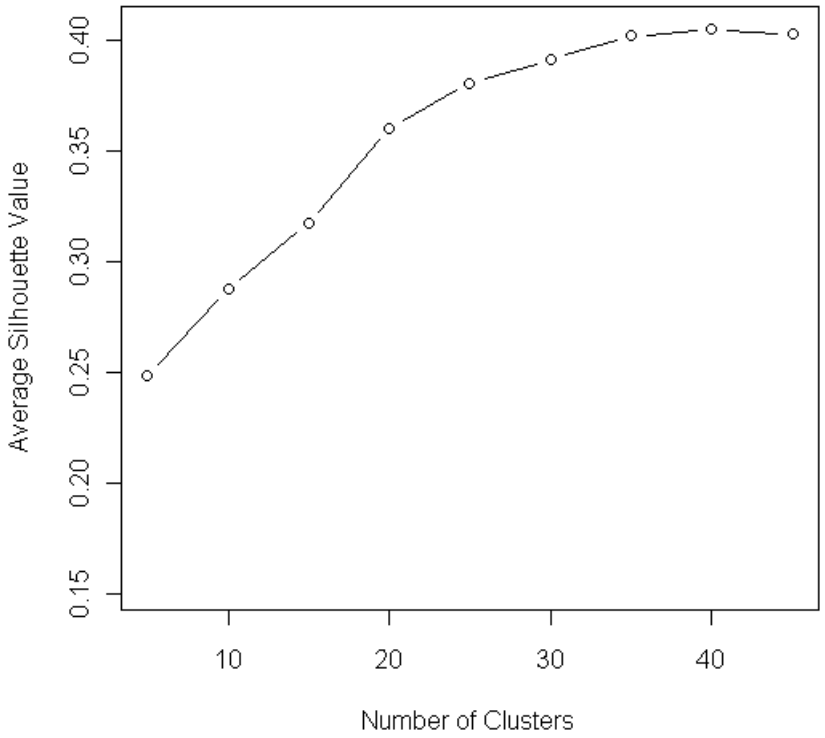

Figure 37 Average Silhouette Values for the TREC Data

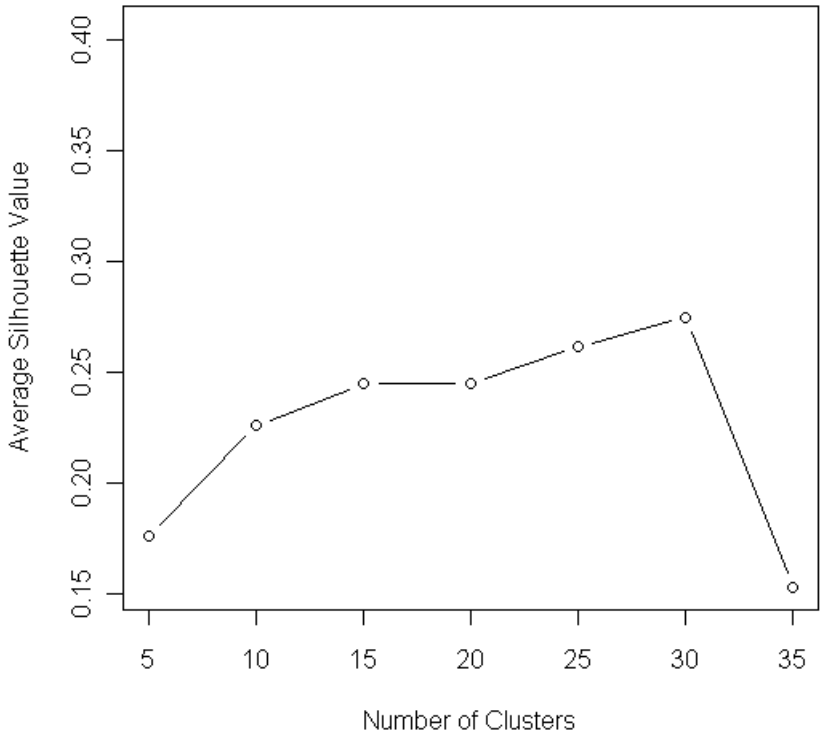

Figure 38 Average Silhouette Values for the CEAS Data

Table 15 TREC Message Counts for Random Boost Evaluation and Table 16 CEAS Message Counts for Random Boost Evaluation provide descriptions of the TREC and CEAS data, broken down by the number of spam and not-spam messages. The messages in the "Sample" and "Medoids" columns are simply a subset of the messages in the "Train" column, while the "Test" data is a disjoint (hold out) set.

Table 15 TREC Message Counts for Random Boost Evaluation

	Total	Train	Sample	Medoids	Test
Spam	50,199	6,021	1,989	28	44,178
Not Spam	25,220	1,521	511	12	23,699
Total	75,419	7,542	2,500	40	67,877
Prob(Spam)	66.56%	79.83%	79.56%	70.00%	65.09%

Table 16 CEAS Message Counts for Random Boost Evaluation

	Total	Train	Sample	Medoids	Test
Spam	110,576	11,020	1,960	25	99,556
Not Spam	27,129	2,751	540	5	24,378
Total	137,705	13,771	2,500	30	123,934
Prob(Spam)	80.30%	80.02%	78.40%	83.33%	80.33%

"Leave One Out" cross validation was used with the labeled medoids to find the optimal parameters for each of the learning algorithms; e.g., the number of stumps for Logit Boost, the number of features and trees for Random Forest, the number of features for Random Project, and the number of features and stumps for Random Boost.

5.3.4 Performance Evaluation

The Figure 39 TREC ROC Curves for Random Boost Evaluation and Figure 40 CEAS ROC Curves for Random Boost Evaluation graphs compare the Receiver

Operating Characteristic (ROC) curves for the Random Forest, Random Boost, Logit Boost, and Random Project algorithms on the TREC and CEAS data sets.

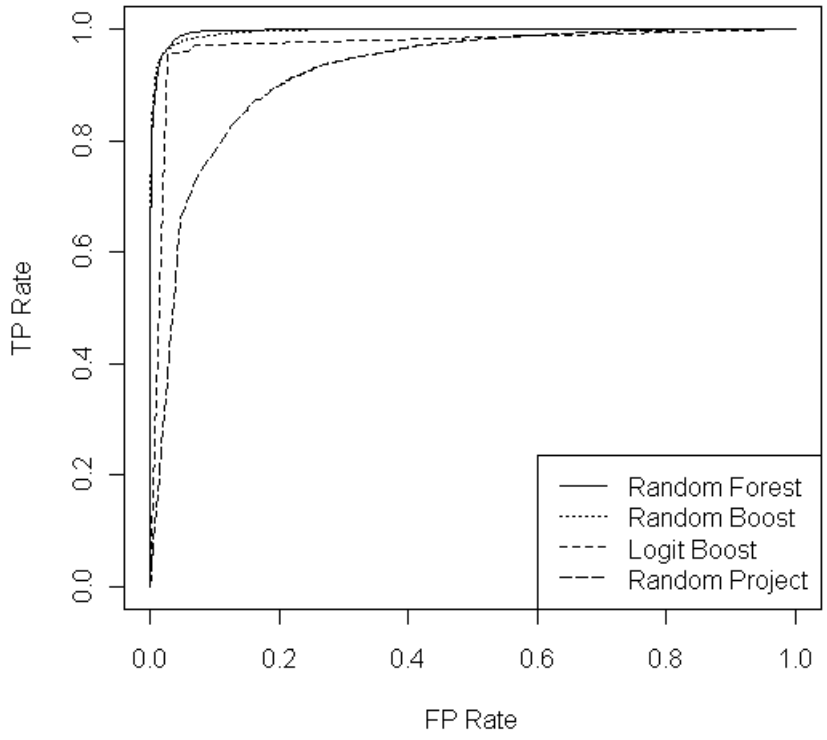

Figure 39 TREC ROC Curves for Random Boost Evaluation

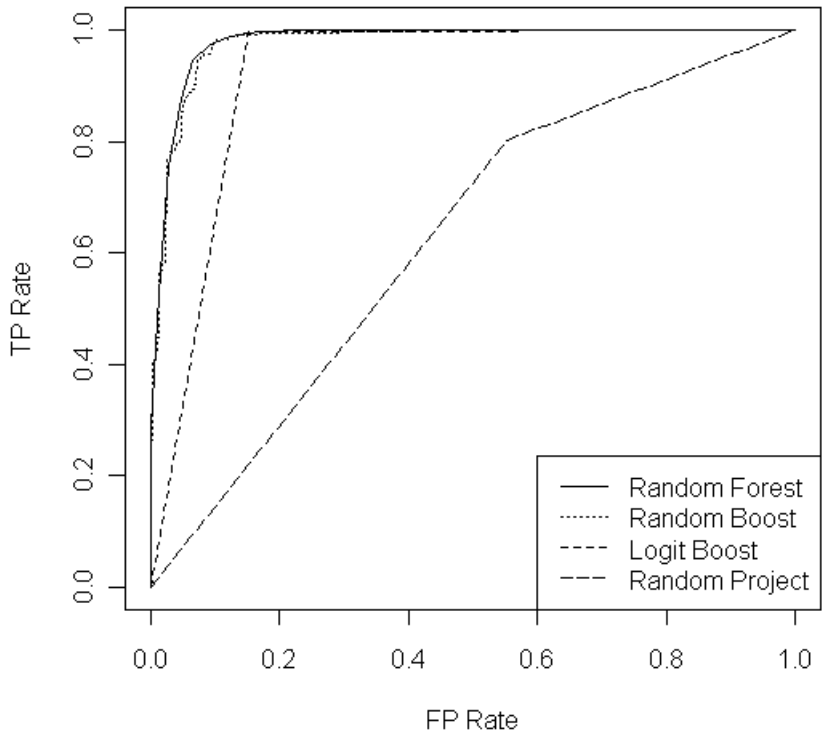

Figure 40 CEAS ROC Curves for Random Boost Evaluation

Table 17 TREC Performance Comparison for Random Boost Evaluation and Table 18 CEAS Performance Comparison for Random Boost Evaluation compare the performance of the four learning approaches in terms of Train Complexity, Test Complexity, the Area Under the ROC Curve (AUC) and 95% Confidence Interval (CI) Half Width for the TREC and CEAS data sets. The Train Complexity is the number of features that had to be evaluated during training: a product of the number of split points and the number of features evaluated for each split point. The Test Complexity is the average number of feature evaluations necessary to classify a new message. For the Random Forest algorithm, this is the number of decision nodes processed instead of the number of trees. For the Logit Boost, Random Boost, and Random Project algorithms,

this is the number of stumps in the ensemble. The AUC is the probability that the model assigns a larger Probability(spam | message, model) to a randomly selected spam message than a randomly selected not-spam message. The 95% Confidence Interval Half Width is the value subtracted from the AUC to derive the lower bound of a 95% confidence interval for the AUC and the value added to the AUC to derive the upper bound of a 95% confidence interval for the AUC.

Table 17 TREC Performance Comparison for Random Boost Evaluation

Model	Train Complexity	Test Complexity	AUC	95% CI Half Width
Random Forest	1370 * 12	693	99.64%	0.17%
Random Boost	150 * 12	150	99.57%	0.19%
Logit Boost	100 * 3843	100	97.17%	0.48%
Random Projection	100 * 2000	100	92.22%	0.78%

Table 18 CEAS Performance Comparison for Random Boost Evaluation

Model	Train Complexity	Test Complexity	AUC	95% CI Half Width
Random Forest	303 * 12	199	97.92%	0.41%
Random Boost	50 * 12	50	97.66%	0.43%
Logit Boost	100 * 4063	50	92.22%	0.77%
Random Projection	100 * 2000	100	62.44%	1.39%

The difference in AUC between the Random Forest and Random Boost algorithms is not statistically significant. Random Boost is similar in performance to the Random Forest algorithm, but at $1/4^{th}$ of the test complexity and $1/6^{th}$ of the training complexity. The Figure 41 TREC AUC Confidence Intervals for Random Boost Evaluation and Figure 42 CEAS AUC Confidence Intervals for Random Boost Evaluation graphs provide a graphical comparison of the confidence intervals for the 4 algorithms. The Random Project algorithm does not appear on the second graph because it performs much worse than the other algorithms.

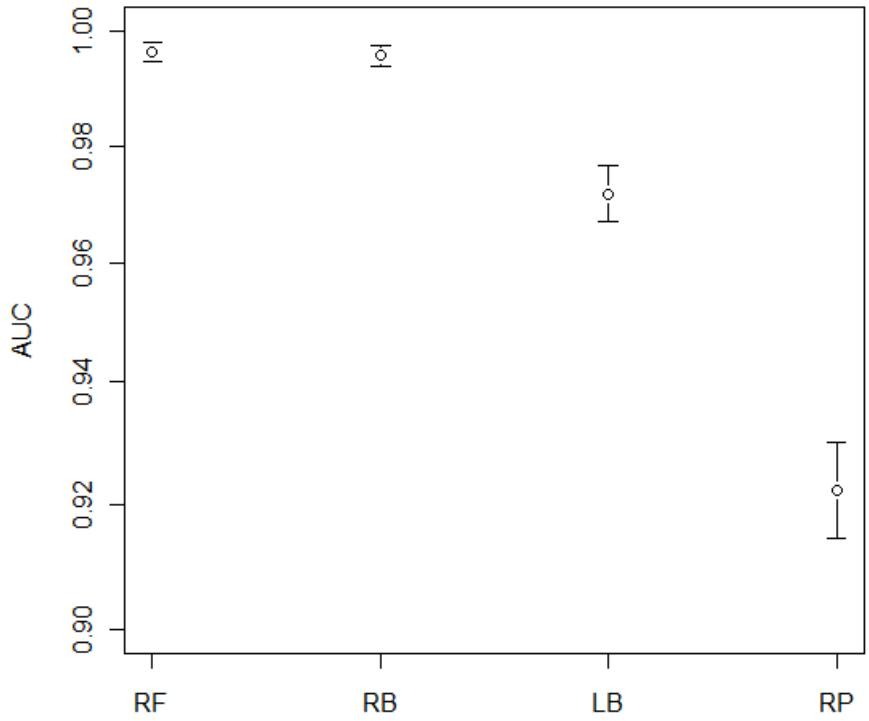

Figure 41 TREC AUC Confidence Intervals for Random Boost Evaluation

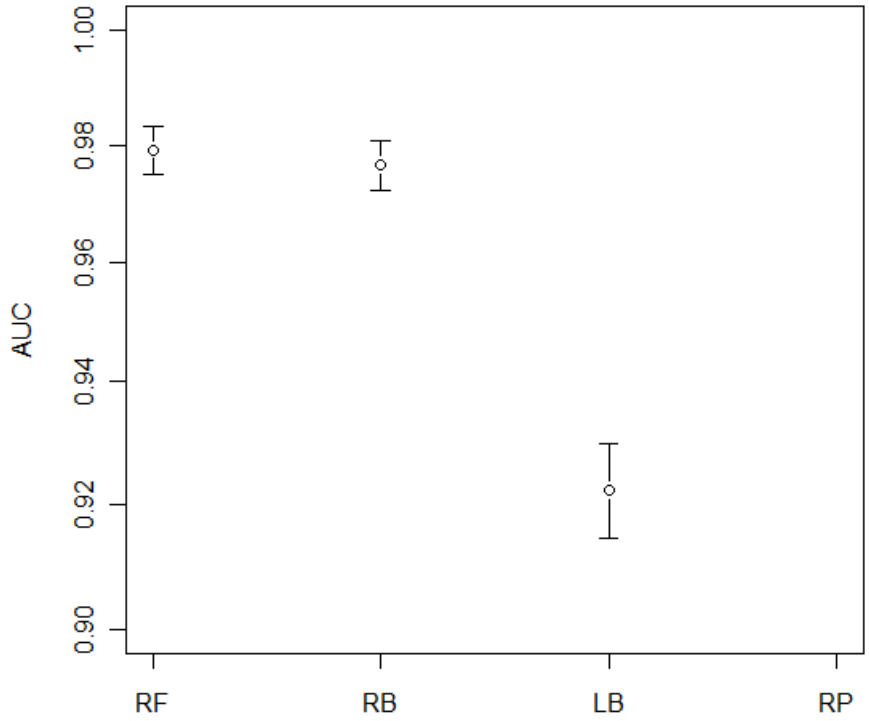

Figure 42 CEAS AUC Confidence Intervals for Random Boost Evaluation

The AUC can be interpreted as the probability that a randomly selected spam message will be assigned a larger Probability(spam | message, model) than a randomly selected not-spam message. The 95% confidence interval for the AUC is based on the maximum variance for all possible distributions where the expected value is equal to the observed AUC, and Chebychev's inequality [63]:

Equation 5.9 Radius for AUC Confidence Interval

$$half\ width\ of\ 95\%\ confidence\ interval\ for\ AUC$$

$$= \sqrt{\frac{20 * AUC * (1 - AUC)}{\min\left(count(Spam), count(NotSpam)\right)}}$$

The Figure 41 TREC AUC Confidence Intervals for Random Boost Evaluation and Figure 42 CEAS AUC Confidence Intervals for Random Boost Evaluation graphs show the confidence intervals for the Random Boost algorithm and the Random Forest algorithm overlap substantially, implying that the classification performance of the Random Boost algorithm is substantially similar to the Random Forest algorithm while the Random Boost algorithm is approximately 4 times as fast as the Random Forest algorithm. The Figure 41 TREC AUC Confidence Intervals for Random Boost Evaluation and Figure 42 CEAS AUC Confidence Intervals for Random Boost Evaluation graphs also show the confidence intervals for the Random Boost and the Logit Boost algorithms do not overlap, implying the performance of the Random Boost algorithm is substantially better than the performance of the Logit Boost algorithm. For the TREC data, the Random Boost algorithm takes 50% longer than Logit Boost to classify messages; but for the CEAS data, the Random Boost algorithm is just as fast as the Logit Boost.

The Logit Boost algorithm uses only 18 out of 3,843 possible features for the TREC data set and even fewer (4 out of 4,063 possible features) for the CEAS data set. When comparing the performance of the Random Boost and Logit Boost algorithms, the fewer the features used by Logit Boost the greater the performance increase provided by Random Boost. Logit Boost used the same few features for many stumps; e.g. "unsubscribe" appears in several of the 100 stumps for the TREC model. In fact, the average number of stumps per feature for Logit Boost model for the TREC data was 100/18=5.6. Meanwhile, Random Boost makes use of more features (135 for the TREC

data set and 50 for the CEAS data set). It should be noted that the Random Boost algorithm made use of additional feature for the CEAS data set, without increasing the test complexity (compared to Logit Boost).

The Figure 43 TREC ROC Curves for Random Boost Evaluation (logarthmic scaling) and Figure 44 CEAS ROC Curves for Random Boost Evaluation (logarithmic scaling) graphs show the details of differences in performance for lower False Positive (FP) rates by using logarithmic scaling for the False Positive (FP) Rate and True Positive (TP) Rate axes. The superior performance of Random Boost compared to Random Forest for CEAS looks remarkable. There may be thus an advantage for Random Boost in terms of supporting a finer granularity of distinction (each Random Boost stump increases/decreases the numeric log odds, while each Random Forest tree simply casts a vote for a classification label).

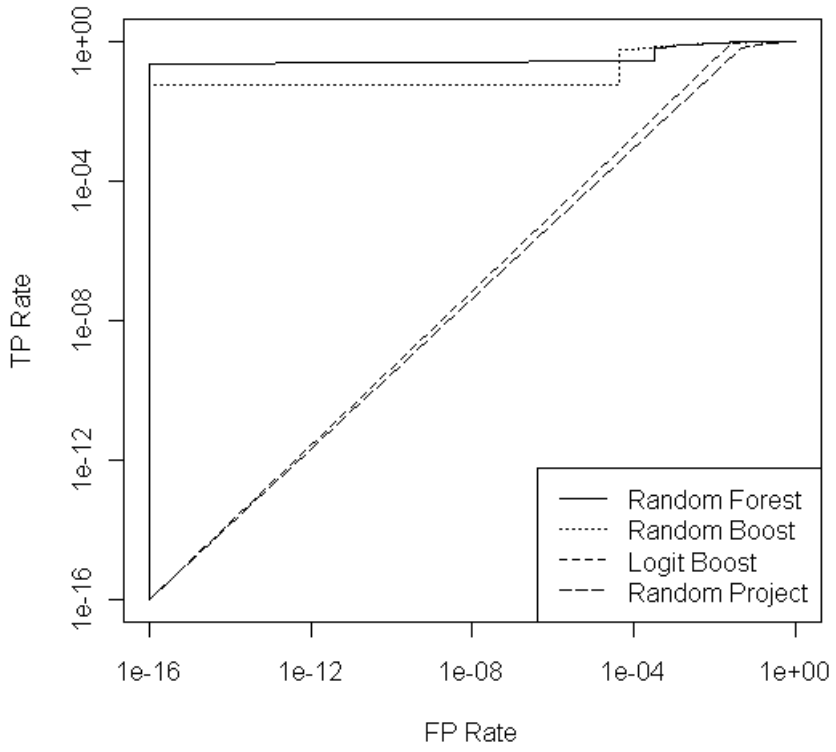

Figure 43 TREC ROC Curves for Random Boost Evaluation (logarthmic scaling)

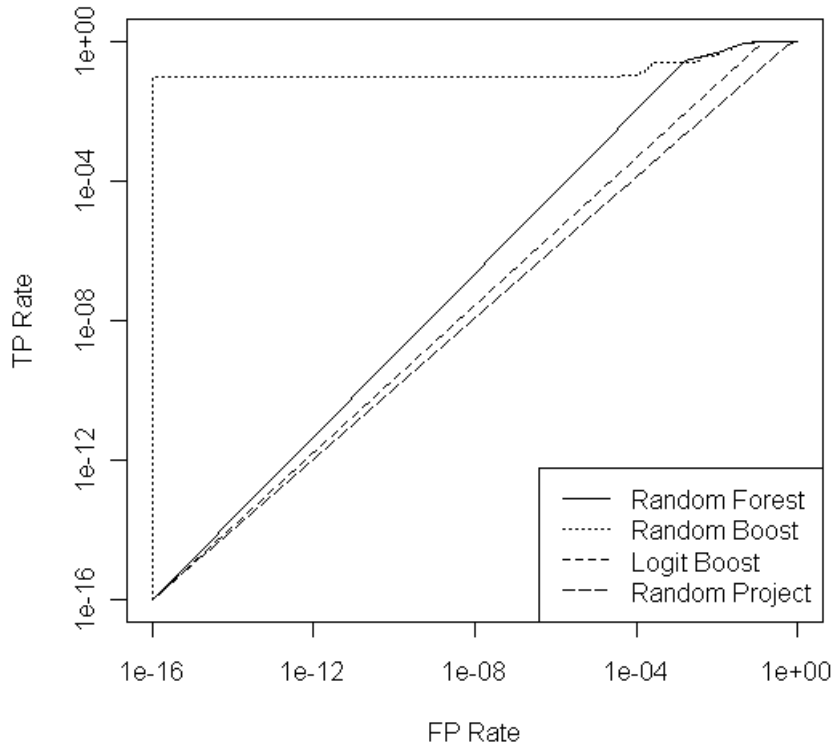

Figure 44 CEAS ROC Curves for Random Boost Evaluation (logarithmic scaling)

Random Boost also takes less time to train than the other methods. Table 17 TREC Performance Comparison for Random Boost Evaluation and Table 18 CEAS Performance Comparison for Random Boost Evaluation compare the training complexity of the four methods for the two data sets. For the TREC data, Random Boost evaluated only 12 randomly selected features at a time to find the best feature for each of its 150 decision stumps (1,800 feature evaluations), while Logit Boost evaluated all 3,843 features to find the best feature for each of its 100 decision stumps (384,300 feature evaluations). Random Forest evaluated 12 randomly selected features at a time to construct each of the 1,370 decision nodes (16,440 feature evaluations); and Random Projection evaluated 2,000 features to find the best feature for each of its 100 regression

stumps (200,000 feature evaluations). For the CEAS data, Random Boost evaluated only 12 randomly selected features at a time to find the best feature for each of its 50 decision stumps (600 feature evaluations), while Logit Boost evaluated all 4,063 features to find the best feature for each of its 100 decision stumps (406,300 feature evaluations). Random Forest evaluated 12 randomly selected features at a time to construct each of the 303 decision nodes (3,636 feature evaluations); and Random Projection evaluated 2,000 features to find the best feature for each of its 100 regression stumps (200,000 feature evaluations).

This research proposed two alternative forms of random projections and compared their performance for robust and efficient spam detection filters, which are trained using a small number of examples. Robustness refers to learning and adaptation that leads to a high level of performance despite data variability, while efficiency is concerned with (i) the complexity of the detection method employed; and (ii) and the amount of resources used for training and retraining. The *Random Project* method employs a random projection matrix to produce linear combinations of input features, while the *Random Boost* method employs random feature selection to enhance the performance of the Logit Boost algorithm. Random Boost is, in fact, a combination of *Logit Boost* and *Random Forest.* Experimental results, using TREC and CEAS as challenging spam application domains, show that the Random Boost method significantly improves the performance of the spam filter compared to the Logit Boost algorithm (e.g., a 5% increase in AUC, which is the area under the Receiver Operating Characteristic

curve), and yields similar classification accuracy compared to the Random Forest method but using only one fourth the runtime complexity of the Random Forest algorithm.

In application domains, such as spam detection, where obtaining labeled training examples is difficult/expensive and one has to further contend with ever changing data streams, one thrives to produce the best classifier possible without requiring large amounts of labeled training examples. The Logit Boost algorithm uses a greedy approach to learning, focusing on using the few best features for distinguishing spam from not-spam. The Random Forest algorithm generates significantly more accurate spam detection models than Logit Boost by using many features for classifying new messages, but the Random Forest models are significantly more expensive to use for classifying new messages. The proposed Random Project approach employs random projections (linear combinations of the input features) with the Logit Boost learning algorithm to generate spam detection models, while the proposed Random Boost approach combines the Logit Boost and Random Forest learning approaches, by incorporating random feature selection as part of the Logit Boost learning algorithm. The Random Project approach fails to provide better accuracy or faster message classification, compared to the Logit Boost method, though it was faster for training. The weak performance of the Random Project method is the result of combining features found in spam messages with features found in not-spam messages. In future work, it may be useful to constrain the linear combinations for Random Project so that spam features are only combined with other spam features and not-spam features are only combined with other not-spam features. The Random Boost method, on the other hand, significantly improves the

accuracy of the spam detection model with little or no additional cost for message classification compared Logit Boost.

A robust spam detection model is expected to withstand noise in the distribution of features used to distinguish spam from non-spam messages. Robustness is thus predicated on the ability of the spam detection model to cope with the "noise" injected by the spammer. This is characteristic of adversarial behavior with "noise" characterized by misleading, confusing, and continuously changing contents. For example, spammers would obfuscate words like "free" (f-r-e-e) or add chaff, such as text from news articles.

Making use of additional features for classification makes the Random Boost algorithm a robust alternative to the Logit Boost algorithm. It offers similar classification accuracy to the Random Forest algorithm with only a fraction of the runtime cost. Next steps for the Random Boost algorithm include evaluating the performance of the algorithm in the context of an online learning environment with active adversarial response, incorporating unlabeled data to enhance generalization performance, and taking advantage of additional training strategies, e.g., transfer learning, co-training, and multi-task learning.

5.4 Adversarial Learning

In public e-mail systems, it is possible to solicit annotation help from users to train spam detection models. For example, we can occasionally ask a selected user to annotate whether a randomly selected message intended for their inbox is spam or not spam. Unfortunately, it is also possible that the user being solicited is an internal threat and has malicious intent. Similar to an adversary, such a user may want to introduce

noise: to confuse the spam classifier into believing a spam message is not spam (to ensure delivery of similar messages), or to confuse the spam classifier into believing a non-spam message is spam (to prevent delivery of similar messages). Inspired by the Randomized Hough-Transform (RHT), a set of Support Vector Machines (SVMs) is trained from randomly chosen data subsets to vote to identify training examples that have been mislabeled. The labels for messages which on the average appear on the wrong side of the decision boundary are flipped and a final SVM model is trained using the modified labels. Two data sets are used for evaluating the proposed RHT-SVM method: the TREC 2007 Spam Track data and the CEAS 2008 Spam data. To preserve the time ordered nature of the data stream, for each of the data sets, the first 10% of the messages are used for training, and the remaining 90% of the messages are used for evaluation. Separate adversarial experiments are conducted for flipping spam labels and non-spam labels. For 10 iterations, labels are flipped for a randomly selected subset of 5% of the training data and the final RHT-SVM is evaluated on the test set. Performance of the RHT-SVM is compared to the performance of the state of the art Reject On Negative Impact (RONI) algorithm. RHT-SVM shows an average 9.3% increase in the F measure compared to RONI (99.0% versus 90.6%), as well as significant improvements in other evaluation metrics. The flip sensitivity for RHT-SVM is 95.9% and the flip specificity is 99.0%. It also takes over 90% less time to complete the RHT-SVM experiments compared to the RONI experiments (20 minutes per experiment instead of 360 minutes).

It is often difficult to obtain annotated data for pattern recognition tasks; however, public e-mail service providers have the ability to solicit annotation support from their

users. A select set of users can be asked to occasionally provide a class label for a randomly selected incoming e-mail message. This, of course, allows an adversary to taint the data used to train the spam detection model. An adversary might mislabel a spam message as not spam in order to allow similar spam messages to be delivered in the future. Alternatively an adversary might mislabel a non-spam message as spam in order to prevent similar message from being delivered in the future. Either of these alternatives compromises the integrity of the spam detection model. While it may be possible to restrict invitations for annotations to well-established accounts, an adversary may create accounts with the intent to influence the training of the spam detection model. Even under the best of circumstances, noise (errors) may be present in the annotated labels as annotators may unintentionally make errors (as seen in earlier work with Social Network Analysis features). We propose a new method for ensuring the integrity of the annotated labels for an e-mail message corpus, based on repeated estimation of the distance between an observation and the decision boundary.

5.4.1 Background

For security applications where an adversary wants to avoid detection, the distribution of data for the test environment may not be the same as the distribution of data for the training environment. An adversary may manipulate the features used for representing observations or the labels used for training detection models. In [64], a taxonomy is presented describing possible attacks against machine learning systems. In [65], the Naïve Bayes learning algorithm was modified to make the resulting classifier

more robust to adversarial message manipulation, assuming the adversary had perfect knowledge of the training data. In [40], the Adversarial Classifier Reverse Engineering (ACRE) framework was introduced to study how an adversary can learn to defeat a spam detection algorithm. In [66], the regularization term of a Support Vector Machine (SVM) learning algorithm was modified to avoid placing too much emphasis on the few best features for classification, because an adversary with access to a classifier can defeat a classifier by manipulating features. In [67], the effects of adding noise to training labels was studied, with an ensemble based approach shown to outperform non-ensemble based approaches. In [68], two algorithms were presented to address missing and corrupted features for the test environment. In [69], the effect of flipping labels on SVM performance was evaluated. In [70], the Reject On Negative Impact (RONI) algorithm was shown to reduce the impact of adversarial feature noise on the learning process.

The Reject On Negative Impact (RONI) approach can also be applied to adversarial label noise. The goal of the RONI algorithm is to determine which observations in the training set should be rejected, i.e. removed from the training set. For each training set observation Q, a 20 message training set **T** and a 50 message validation set **V** is sampled from the training data. Both **T** and $\mathbf{T} \cup Q$ are used for training (with/without observation Q), with the change in classification performance averaged over 5 trials. If classification results are more accurate without the observation Q, it is removed from the training set.

Support Vector Machines for pattern classification were introduced in [71], with an extension proposed in [72]. Given a set of training examples

$\{(\mathbf{x}_1, \mathrm{y}_1), (\mathbf{x}_2, \mathrm{y}_2), ..., (\mathbf{x}_n, \mathrm{y}_n)\}$, with $\mathbf{x}_i \in R^p$ and $y_i \in \{-1, +1\}$, a ν support vector classifier is derived by solving the following optimization problem:

$$\min_{\boldsymbol{\alpha}} \frac{1}{2} \boldsymbol{\alpha}^T \mathbf{Q} \boldsymbol{\alpha}$$

subject to

$$0 \leq \alpha_i \leq 1$$

$$\sum_{i=1}^{n} \alpha_i = \nu n$$

$$\mathbf{y}^T \boldsymbol{\alpha} = 0$$

where

$$\mathbf{Q}_{i,j} = y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

This is similar to the C support vector classifier discussed in the chapter on machine learning methods. The ν parameter is similar to the C parameter, in that it controls the complexity of the model. The ν parameter provides an upper bound on the training set error, as well as a lower bound on the proportion of the training set to be identified as support vectors. The resulting decision function is:

$$\hat{y}_{new} = sign\left(\sum_{i=1}^{n} \left(y_i \alpha_i K(\mathbf{x}_{new}, \mathbf{x}_i)\right) + b\right)$$

The kernel function $K()$ measures the similarity of two observations. Typical choices for a kernel function include a linear kernel function such as a dot product, or a

non-linear kernel function such as a radial basis function. A non-zero α_i coefficient identifies a training set observation that has been selected as a support vector.

In the case of a linear dot product kernel, the decision function simplifies to:

Equation 5.10 SVM Decision Function

$$\hat{y}_{new} = sign\left(\mathbf{w}^T \mathbf{x}_{new} + b\right)$$

where

$$\mathbf{w} = \sum_{i=1}^{n} y_i \alpha_i \mathbf{x}_i$$

The SVM is a maximum margin classifier, meaning the decision boundary is chosen to maximize the minimum distance between each observation and the decision boundary. For the linear kernel, the minimum signed distance between an observation and the decision boundary is given by:

Equation 5.11 Signed Distance to the Decision Boundary

$$d = \frac{\mathbf{w}^T \mathbf{x}_{new} + b}{\|\mathbf{w}\|}$$

The value of Equation 5.11 Signed Distance to the Decision Boundary is positive when the exemplar appears on the positive side of the boundary and negative otherwise. The value $\frac{1}{\|\mathbf{w}\|}$ is known as the margin of the classifier. Observations with minimum distance to the decision boundary less than the margin are considered to be support vectors.

Conventional smoothing techniques use as much of the data as possible to obtain an initial solution model and then attempt to eliminate misfit data points. RANSAC [73], characteristic of robust statistical methods, starts with a small but randomly chosen data set and estimates some parametric model that best fits the selected data. It then iteratively enlarges the set with consistent data when possible and re-estimates the model and its error. Starting from different subsets, RANSAC derives competing models and chooses the one that yields the smallest error. RANSAC is one of the motivations for the Hough Transform (HT).

The Hough Transform determines for every data point the parameter subspace of models it supports, and increases the votes in the Hough space for all these models. An extension of this principle is to vote for sets of data points instead of single points. The subspace of supported models is then smaller, while the number of different point sets is larger. For example, a hyper plane in R^N is specified by N points, and a single point imposes an $N-1$ dimensional subspace of supported models in the Hough space. When a pair of points is considered, the set of supported models is an $N-2$ dimensional subspace. The voting process is then faster, but we have to consider different pairs instead of only N points. The limiting case is when precisely sets of N points are selected, which then results in a single point in the Hough space. This is the principle of the Randomized Hough Transform (RHT) [74]. Instead of the total number of possible sets of points, only a small number of random sets is selected which is sufficient to find the best model. RHT uses randomly chosen subsets of tokens to vote for final structures, and reduces the

time and storage required compare to the HT. This leads to the novel RHT-SVM method, which is described next.

5.4.2 Proposed Method

As demonstrated in [67], ensembles have been shown to exhibit robustness against noise. While individual members of the ensemble may over fit the training data (exhibiting high variance in the decision boundary), an ensemble reduces variance by generating smoothed (lower variance) estimates of the decision boundary. The Randomized Hough Transform – Support Vector Machine (RHT-SVM) leverages multiple views of the decision boundary to identify observations that have been mislabeled.

The RHT-SVM uses the product of the actual classification label and the average signed distance of an observation from the decision boundary to determine if a training observation has been mislabeled:

Equation 5.12 Expected Geometric Margin

$$f_i = y_i \frac{\sum_{j=1}^{N} \frac{\mathbf{w}^T \mathbf{x}_i + b}{\|\mathbf{w}\|}}{N}$$

where N is the number of iterations of resampling.

The RHT-SVM algorithm is shown in the Figure 45 RHT-SVM Learning Algorithm. The estimate of distance to the decision boundary is not influenced by the

presence of the observation in the training set. In our experiments, T was chosen to be 10 and τ was chosen to be 0.

for iteration = 1 to T
  Randomly divide the training set into 3 partitions.
  for P = 1 to 3
    Construct an SVM using the training data without partition P
      to estimate the distance to the decision boundary for observations in partition P.
Average the distance to the decision boundary for observations in the training set.
Flip the class label for training observations if $f_i < \tau <= 0$.
Train the final RHT-SVM using the modified class labels.

Figure 45 RHT-SVM Learning Algorithm

5.4.3 Experimental Design

Two data sets were used for comparing the performance of the RHT-SVM to the performance of RONI, as well as the performance of an SVM trained on the tainted training data. The TREC 2007 data set [7] [54] consists of 75,419 messages collected during Apr 8 – Jul 6 2007, while the CEAS 2008 data set [75] consists of 137,705 messages collected during Aug 5 – Aug 9 2008. For both data sets, the first 10% of the time ordered e-mail stream was used for training with the remaining 90% of the data used for testing. The Table 19 TREC Message Counts for the RHT-SVM Evaluation show the

message counts for the TREC data set. The proportion of spam is 67% for the TREC data set.

Table 19 TREC Message Counts for the RHT-SVM Evaluation

	Training	Testing	Total
Spam	6,021	44,178	50,199
Not Spam	1,521	23,699	25,220
Total	7,542	67,877	75,419

The Table 20 CEAS Message Counts for RHT-SVM Evaluation show the message counts for the CEAS data set. The proportion of spam is 80% for the CEAS data set.

Table 20 CEAS Message Counts for RHT-SVM Evaluation

	Training	Testing	Total
Spam	11,020	99,556	110,576
Not Spam	2,751	24,378	27,129
Total	13,771	123,934	137,705

The messages for both data sets were preprocessed using the LingPipe library [14] as follows:

- Message text was converted to lower case characters
- Messages were divided into tokens by breaking on spaces, tabs, and punctuation (‘.’, ‘!’, ‘?’)
- English stop words were removed; e.g. articles (‘a’, ‘an’, the’), conjunctions (‘and’, ‘or’), and prepositions (‘for’, ‘in’)
- Tokens were stemmed using the Porter stemmer [15]

The alphanumeric tokens from the training set were then used to derive Term Frequency – Inverse Document Frequency (TF-IDF) representation [47] for the training and testing sets.

The TREC training corpus is comprised of 3,478 distinct tokens, while the CEAS training corpus is comprised of 3,621 distinct tokens.

The linear dot product kernel was selected as the kernel of choice for the SVMs, because it outperformed the non-linear Radial Basis Function (RBF) kernel during 10 fold cross validation of the training data. The only SVM parameter to be optimized was ν, which places an upper bound on training set error and a lower bound on the proportion of training set observations selected as support vectors. Ten fold cross validation of the training set was used to select this value.

Separate experiments were conducted for flipping spam labels and non-spam labels. For 10 iterations, labels were flipped for a randomly selected subset of 5% of the training data. A total of 8 metrics were used to evaluate the performance of the models:

1. Area Under the Receiver Operating Characteristic (ROC) Curve (AUC) [51]: the probability that a randomly selected message from the spam class will be

viewed as more likely to be a spam message than a randomly selected member of the not-spam class

2. Accuracy: the probability that the predicted class for a randomly selected message is the actual class for that message
3. Precision: the probability that a predicted spam message is actually a spam message
4. Recall: the probability that an actual spam message is predicted to be a spam message
5. F Measure: the harmonic mean of Precision and Recall
6. Flip Sensitivity: the probability that a message whose label was flipped is identified as a mislabeled message
7. Flip Specificity: the probability that a message whose label was not flipped is identified as a correctly labeled message
8. nSV: the number of training observations used to define the decision boundary

5.4.4 Performance Evaluation

Table 21 RHT-SVM Performance for Untainted Data shows the test performance of SVMs constructed using the untainted training data. These results are competitive with state of the art performance on these data sets.

Table 21 RHT-SVM Performance for Untainted Data

	TREC	CEAS
AUC	99.99%	99.81%
Accuracy	98.85%	98.98%
Precision	99.19%	99.41%
Recall	99.05%	99.31%
F Measure	99.12%	99.36%
nSV	512	802

Figure 46 TREC ROC Curves for Flipping Positive Labels to Negative shows example ROC curves for the 3 methods being evaluated when tainting positive class training examples of the TREC data set (flipping spam labels to not-spam labels):

- RHT-SVM: SVMs trained using the labels produced by the Randomized Hough Transform – Support Vector Machine algorithm
- Tainted: SVMs trained using the tainted labels
- RONI: the Reject On Negative Impact (RONI) algorithm

The ROC curve is produced by varying the classification threshold from positive infinity to negative infinity, recording the False Positive Rate and the True Positive Rate for each possible threshold. The False Positive Rate is the probability that a randomly selected not-spam message is misclassified as spam, while the True Positive Rate is the probability that a randomly selected spam message is correctly classified as spam.

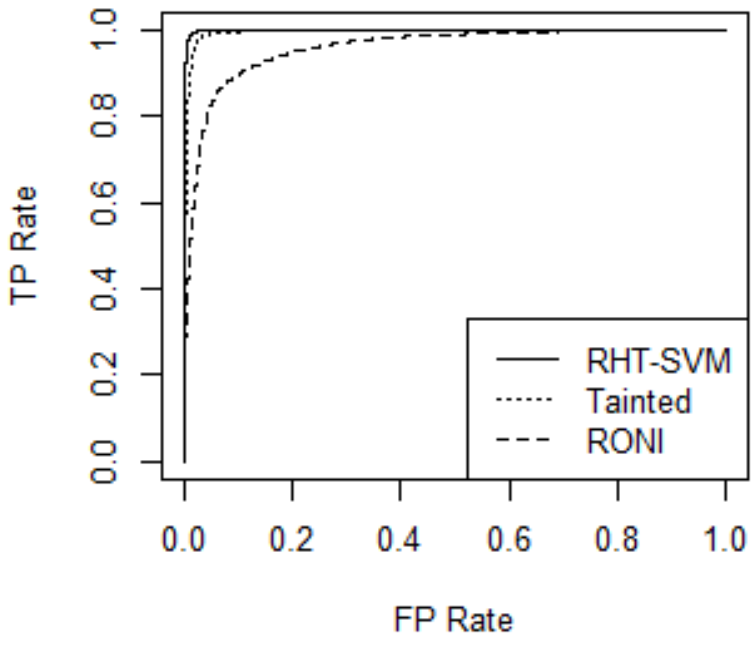

Figure 46 TREC ROC Curves for Flipping Positive Labels to Negative

Table 22 RHT-SVM TREC Performance for Flipping Positive Labels to Negative shows the mean evaluation metrics for the 10 trials when tainting positive class training examples of the TREC data set (flipping spam labels to not-spam labels). The standard error for each of the metrics is shown in parentheses.

Table 22 RHT-SVM TREC Performance for Flipping Positive Labels to Negative

	Tainted	RHT-SVM	RONI
AUC	99.09% (0.10%)	99.84% (< 0.01%)	96.08% (0.27%)
Accuracy	94.53% (0.39%)	98.69% (0.03%)	87.71% (0.58%)
Precision	99.00% (0.05%)	99.18% (0.01%)	97.12% (0.33%)
Recall	92.53% (0.63%)	98.81% (0.04%)	83.62% (1.08%)
F Measure	95.64% (0.33%)	98.99% (0.02%)	89.82% (0.55%)
Flip Sensitivity	0.00% (0.00%)	99.60% (0.11%)	21.11% (1.07%)
Flip Specificity	100.00% (0.00%)	99.38% (0.02%)	99.27% (0.03%)
nSV	1,497.2 (7.2)	533 (4.2)	1741.5 (60.1)

Figure 47 TREC ROC Curves for Flipping Negative Labels to Positive shows example ROC curves for the 3 methods being evaluated when tainting negative class training examples of the TREC data set (flipping not-spam labels to spam labels).

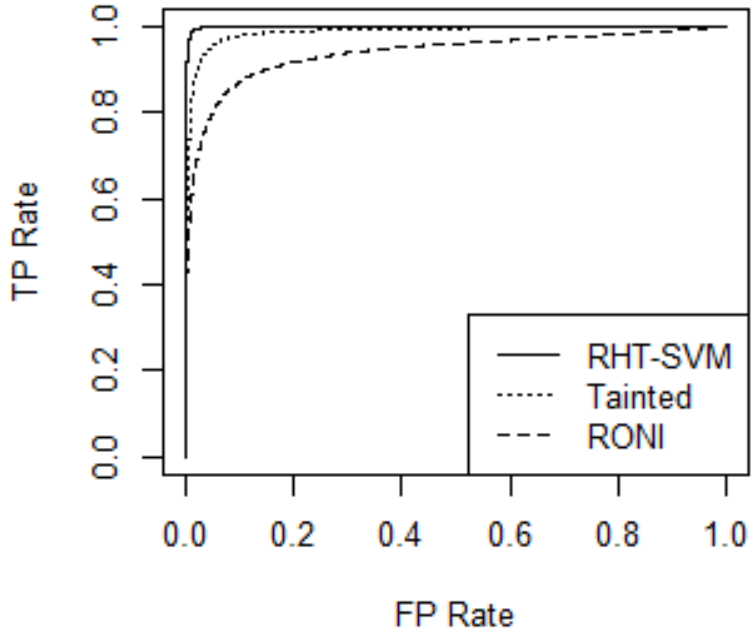

Figure 47 TREC ROC Curves for Flipping Negative Labels to Positive

Table 23 RHT-SVM TREC Performance for Flipping Negative Labels to Positive shows the mean evaluation metrics for the 10 trials when tainting negative class training examples of the TREC data set (flipping not-spam labels to spam labels). The standard error for each of the metrics is shown in parentheses.

Table 23 RHT-SVM TREC Performance for Flipping Negative Labels to Positive

	Tainted	RHT-SVM	RONI
AUC	98.77% (0.07%)	99.76% (0.03%)	94.24% (0.24%)
Accuracy	95.42% (0.18%)	98.52% (0.08%)	86.08% (0.42%)
Precision	95.38% (0.26%)	98.36% (0.08%)	85.52% (0.64%)
Recall	97.70% (0.29%)	99.38% (0.05%)	94.72% (0.47%)
F Measure	96.52% (0.14%)	98.87% (0.06%)	89.86% (0.27%)
Flip Sensitivity	0.00% (0.00%)	95.07% (0.34%)	5.07% (0.36%)
Flip Specificity	100.0% (0.00%)	98.95% (0.05%)	99.78% (0.01%)
nSV	1,121.4 (5.3)	531.9 (3.6)	980.3 (5.2)

Figure 48 CEAS ROC Curves for Flipping Positive Labels to Negative shows example ROC curves for the 3 methods being evaluated when tainting positive class training examples of the CEAS data set (flipping spam labels to not-spam labels).

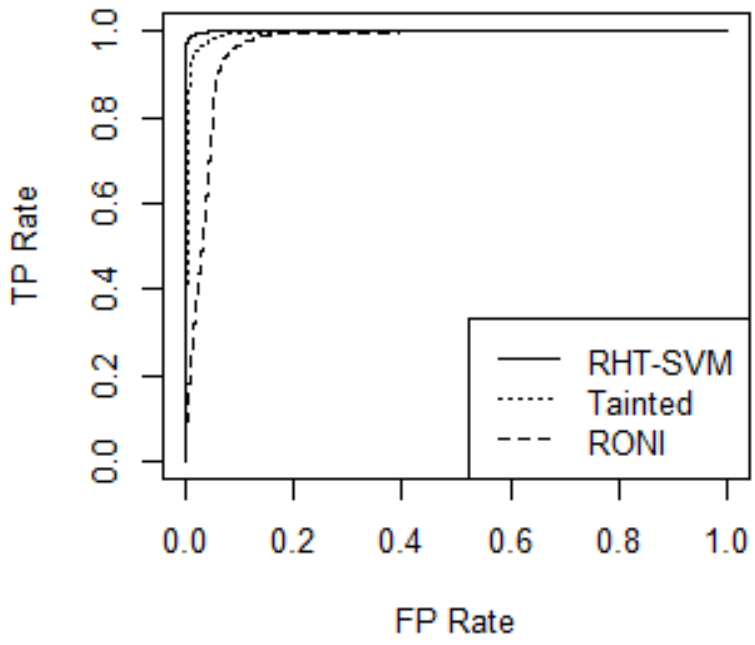

Figure 48 CEAS ROC Curves for Flipping Positive Labels to Negative

Table 24 RHT-SVM CEAS Performance for Flipping Positive Labels to Negative shows the mean evaluation metrics for the 10 trials when tainting positive class training examples of the CEAS data set (flipping spam labels to not-spam labels). The standard error for each of the metrics is shown in parentheses.

Table 24 RHT-SVM CEAS Performance for Flipping Positive Labels to Negative

	Tainted	RHT-SVM	RONI
AUC	99.02% (0.07%)	99.82% (< 0.01%)	95.79% (0.24%)
Accuracy	95.39% (0.25%)	98.95% (0.01%)	83.46% (1.50%)
Precision	99.37% (0.03%)	99.43% (< 0.01%)	98.42% (0.10%)
Recall	94.87% (0.32%)	99.26% (0.02%)	80.70% (1.89%)
F Measure	97.06% (0.17%)	99.34% (0.01%)	88.58% (1.15%)
Flip Sensitivity	0.00% (0.00%)	99.42% (0.08%)	34.38% (0.54%)
Flip Specificity	100.0% (0.00%)	99.29% (0.02%)	97.88% (0.03%)
nSV	2,523.3 (13.9)	849.7 (2.8)	2,094.3 (24.8)

Figure 49 CEAS ROC Curves for Flipping Negative Labels to Positive shows example ROC curves for the 3 methods being evaluated when tainting negative class training examples of the CEAS data set (flipping not-spam labels to spam labels).

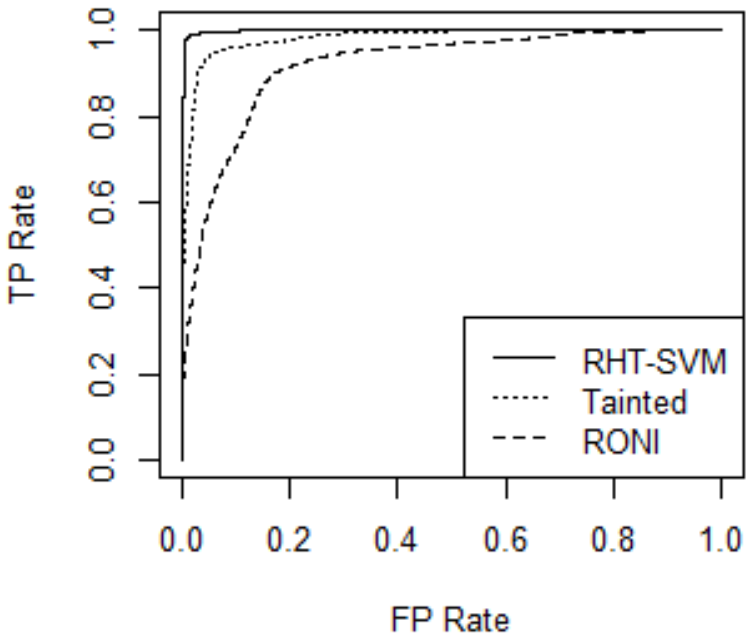

Figure 49 CEAS ROC Curves for Flipping Negative Labels to Positive

Table 25 RHT-SVM CEAS Performance for Flipping Negative Labels to Positive shows the mean evaluation metrics for the 10 trials when tainting negative class training examples of the CEAS data set (flipping not-spam labels to spam labels). The standard error for each of the metrics is shown in parentheses.

Table 25 RHT-SVM CEAS Performance for Flipping Negative Labels to Positive

	Tainted	RHT-SVM	RONI
AUC	98.19% (0.20%)	99.63% (0.03%)	93.08% (0.67%)
Accuracy	94.37% (0.39%)	98.37% (0.09%)	90.61% (0.43%)
Precision	94.61% (0.35%)	98.53% (0.13%)	91.78% (0.46%)
Recall	98.63% (0.26%)	99.46% (0.04%)	97.02% (0.34%)
F Measure	96.57% (0.23%)	98.99% (0.05%)	94.32% (0.25%)
Flip Sensitivity	0.00% (0.00%)	89.48% (0.33%)	3.83% (0.31%)
Flip Specificity	100.0% (0.00%)	98.23% (0.04%)	99.71% (0.01%)
nSV	1,917.6 (8.5)	960.7 (6.0)	1,648.8 (6.4)

For security applications, an adversary may actively attempt to influence the machine learning process. For example, in public e-mail systems, we may occasionally ask a selected user to annotate whether a randomly selected message destined for their inbox is spam or not spam. An adversary may mislabel examples to confuse the spam classifier into believing a spam message is not spam (to ensure delivery of similar messages), or to confuse the spam classifier into believing a non-spam message is spam (to prevent delivery of similar messages). A new method for ensuring the integrity of the annotated labels for an e-mail message corpus was proposed, based on repeated estimation of the distance between an observation and the decision boundary. Inspired by the Randomized Hough-Transform (RHT), a set of Support Vector Machines (SVMs) is trained from randomly chosen data subsets to vote to identify training examples that have

been mislabeled. The labels for messages which appear on the average on the wrong side of the decision boundary are changed and a final SVM model is trained using the modified labels. Two data sets were used for evaluating the proposed method: the TREC 2007 Spam Track data and the CEAS 2008 Spam data.

For all experiments, the Randomized Hough Transform – Support Vector Machine (RHT-SVM) outperformed both the SVM constructed using the tainted data and RONI. The RHT-SVM may also be useful for ensuring the integrity of training labels in other domains. Future work will be focused on evaluating the RHT-SVM in additional domains, as well as evaluating its robustness against adversarial feature noise. While the RHT-SVM approach requires the construction of fewer models (compared to RONI which requires 10 models per training observation), it does require training a set of Support Vector Machines (SVMs) from randomly chosen data subsets to vote to identify training examples that have been mislabeled. We plan to evaluate the use of random subspace projections to reduce computational complexity while possibly making the RHT-SVM more robust to feature noise from external adversaries.

6. PHISHING DETECTION

Phishing is an attempt to steal a user's identity. This is typically accomplished by sending an e-mail message to a user, with a link directing the user to a web site used to collect personal information. Phishing detection systems typically rely on content filtering techniques, such as Latent Dirichlet Allocation (LDA), to identify phishing messages. In the case of spear phishing, however, this may be ineffective because messages from a trusted source may contain little content. In order to handle such emerging spear phishing behavior, we propose as a first step the use of Spectral Clustering to analyze messages based on traffic behavior. In particular, Spectral Clustering is used to analyze the links between URL substrings for web sites found in the message contents. The spectral representation and cluster membership is then used to construct a Random Forest classifier for phishing detection. Data from the Phishing E-mail Corpus and the Spam Assassin E-mail Corpus are used to evaluate this approach. Performance evaluation metrics include the Area Under the receiver operating characteristic Curve (AUC), as well as accuracy, precision, recall, and the (harmonic mean) F measure. Performance of the integrated Spectral Clustering and Random Forest approach is found to provide significant improvements in all the metrics listed, compared to a content filtering technique such as LDA coupled with text message deletion done randomly or in an adaptive fashion using adversarial learning. The Spectral Clustering

approach is robust against the absence of content. In particular, we show that Spectral Clustering yields (99.8%, 97.8%) for (AUC, F measure) compared to LDA that yields (94.6%, 89.4%) and (79.6%, 57.9%) when the content of the messages is reduced to 10% of their original size using random and adversarial deletion, respectively. The difference is most striking at low False Positive (FP) rates.

In order to deliver their messages and collect the victim's information, phishers often use the same infrastructure repeatedly. The Figure 50 Phishing Infrastructure diagram illustrates this behavior. A message is sent from the phisher to a mail server, in order to relay the message to the victim. The user then visits the phisher's web server by clicking a link in the message, prompting the user for their account information. The web server then stores the account information for later retrieval by the phisher.

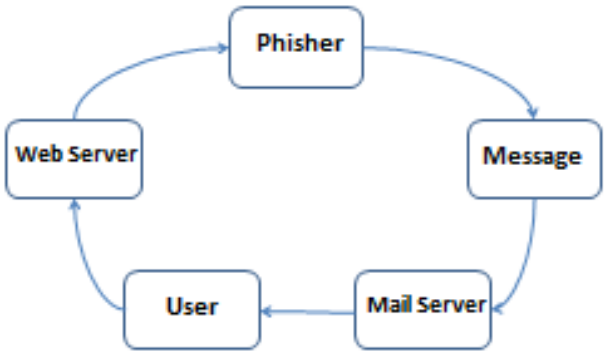

Figure 50 Phishing Infrastructure

By analyzing connections between the web server URLs, provided by the e-mail message, one will be better able to detect spear phishing messages containing very little content. The web server connections can be used to compute a Laplacian matrix, a

matrix representation of the graph consisting of messages and the URLs connecting them. Spectral clustering uses the spectrum (eigenvalues) of the Laplacian matrix to group messages together. Cluster membership, along with the spectral representation of the messages, can then be used to build a classification model for detecting phishing messages.

6.1 Background

Content filtering approaches to phishing detection rely on text contents, where a filter either disallows delivery of messages recognized as phishing messages or delivers suspect messages to a special folder for careful review by the intended recipient. A comparison of machine learning techniques for phishing detection using Term Frequency – Inverse Document Frequency (TF-IDF) representation is found in [76]. Training and test features were a function of the product of how often a term occurs within a document (TF) and the inverse of the proportion of documents containing the term (IDF). This is known as a bag-of-words approach, because relationships between words are not directly considered in this representation. A total of 2,889 e-mails were used for evaluation, with 40.5% of the data set labeled as phishing messages. The phishing messages were composed of 1,171 out of 1,423 messages from the Phishing Corpus [77]. The non-phishing messages came from the researchers' personal mail boxes. Comparing neural network, logistic regression, Bayesian Additive Regression Trees (BART), Classification And Regression Tree (CART), Random Forest, and Support Vector Machine (SVM)

models, the results showed that the Random Forest outperformed the other methods when misclassification errors had equal costs.

In [8], Latent Dirichlet Allocation (LDA) was explored as an improved representation for phishing message detection. LDA estimates the probability of each topic for a message based on the contents of the message, where the topic probabilities sum to one. Given Dirichlet parameters $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, the joint probability of a topic mixture $\boldsymbol{\theta}$, and a set of N topics z and words w, is given by [78]:

Equation 6.1 Joint Probability of a Topic Mixture

$$p(\boldsymbol{\theta}, \mathbf{z}, \mathbf{w} \mid \boldsymbol{\alpha}, \boldsymbol{\beta}) = p(\boldsymbol{\theta} \mid \boldsymbol{\alpha}) \prod_{i=1}^{N} p(z_i \mid \boldsymbol{\theta})\, p(w_i \mid z_i, \boldsymbol{\beta})$$

The Topic Modeling Toolbox [79] was used to discover the distribution of terms for a topic and the distribution of topics for a document, using a collapsed Gibbs sampling approach. LDA topic distributions for phishing and non-phishing topics were used as features for classification. As shown later in this chapter, performance drops when one reduces the message content, as might be the case in spear phishing, e.g., the entire message may be simply reduced to the words “Check this out” with a URL link to the phishing site.

Examples of 6 LDA topics (groups of related tokens) discovered in this research are shown in the Figure 51 Examples of Topics from the Phishing and Non Phishing Corpora.

access, limited, paypal
bank, banking, online
credit, information, union
ebay, target, alt
content-type, content-transfer-encoding, charset
</td>,</table>,</tbody>
helvetica, sans-serif, <p><font
list, linux, @linux.ie

Figure 51 Examples of Topics from the Phishing and Non Phishing Corpora

6.2 Proposed Method

A graph is a simple data structure composed of objects (aka nodes or vertices) and links (aka connections or edges). For example, a set of messages can be viewed as a graph. Each message is a node in the graph, while having similar URLs in the message body can be viewed as a link between messages. The graph formed by these connections can be compactly represented by an n x n Laplacian matrix. There are essentially 4 steps in spectral clustering [80]:

1. Compute the Laplacian matrix **L** to represent the set of messages
2. Derive the spectral decomposition of the Laplacian matrix: eigenvalues **Λ** and eigenvectors **Q**
3. Form the new spectral representation **S** from the eigenvectors of **Q**
4. Cluster the rows of the matrix **S**

A normalized Laplacian matrix ***L*** is simply the affinity matrix ***A*** normalized by the degree matrix ***D***:

$$\mathbf{L} = \mathbf{D}^{-1/2} * \mathbf{A} * \mathbf{D}^{-1/2}$$

where

$$\mathbf{A}[i, j] = \begin{cases} \exp\left(-\frac{\text{distance}(i, j)^2}{2\sigma^2} \right), & i \neq j \\ 0, & i = j \end{cases}$$

$$\mathbf{D}[i, j] = \begin{cases} \sum_{j=1}^{n} \mathbf{A}[i, j], & i = j \\ 0, & i \neq j \end{cases}$$

The distance() measure is shown in the Equation 6.2 Jaccard Distance Between Messages. The σ, kernel width, parameter is chosen as the expected distance between a pair of messages.

The spectral decomposition of the Laplacian matrix ***L*** provides the spectrum (eigenvalues) ***Λ*** and characteristic directions (eigenvectors) ***Q*** of the graph [81]. An *n* x *n* Laplacian matrix can be factored as follows:

$$\mathbf{L} = \mathbf{Q} * \mathbf{\Lambda} * \mathbf{Q}^{-1}$$

where

$$\mathbf{\Lambda}[i, j] := \begin{cases} \text{solution of } \det(\mathbf{L} - \mathbf{\Lambda}[i, j] * \mathbf{I}) = 0, & i = j \\ 0, & i \neq j \end{cases}$$

$$\mathbf{Q}[i,] := \text{solution of } (\mathbf{L} - \mathbf{\Lambda}[i, j] * \mathbf{I}) * \mathbf{Q}[i,] = 0$$

The new representation matrix $\boldsymbol{S}$ is then formed as the eigenvectors associated with the k largest eigenvalues; e.g. all eigenvectors associated with eigenvalues greater than zero. The rows are then normalized so they have length equal to one:

$$\mathbf{S}[i, j] = \frac{\mathbf{Q}[i, j]}{\sqrt{\sum_{j=1}^{k} \mathbf{Q}[i, j]^2}}$$

A clustering algorithm such as Partitioning Around Medoids (PAM) [49] can then be used to identify m groups of similar observations. The number of groups m is chosen as the value of m that results in the largest average "silhouette" value for an observation. A classification model can then be constructed using the following features for each observation:

- cluster: the identity of the assigned cluster
- silhouette: the silhouette value for the observation
- neighbor: the identity of the nearest neighboring cluster
- spectral features: the spectral representation for the observation

In [82], spectral clustering was used to cluster spamming servers based on HoneyPot data [83]. The HoneyPot project routinely exposes e-mail addresses on the Internet to identify the e-mail address harvesters (observed via HTTP Get requests) and the spamming e-mail servers (observed by collecting message traffic). Spectral clustering was then used on this bipartite (harvesters, e-mail servers) graph, to identify interesting groups of servers. For example, the authors discovered that spammers involved in phishing often use different infrastructure from spammers who are not involved in phishing, where phishing

was determined by heuristic keyword classification of the associated e-mail content (e.g. occurrence of the keyword "paypal" in the message was used to classify the message as phishing). Our work also uses spectral clustering but it differs in that the goal is e-mail classification (discriminating phishing and non-phishing messages) rather than server classification, and web server links are analyzed instead of e-mail address links between harvester servers and e-mail servers.

Random Forest classifiers are constructed for both the LDA features approach and the Spectral Clustering features approach. To classify new messages, each tree in the Random Forest classification model casts its vote for a class label: phishing or not phishing. The proportion of votes for the phishing class is the probability that a randomly selected tree would classify the message as a phishing message. This is interpreted as the probability of a message being a phishing message.

Phishers often use web server infrastructure to capture account information from victims. The web server Uniform Resource Locator (URL) links can be found in the body of the e-mail messages. These links are often populated by configurable software used by spammers to automate their operations. For example, the web server link is selected from a pool of available web servers/pages. The software then simply populates a phishing link macro for a template.

The Figure 52 Phishing E-mail Example shows a simple example of an e-mail message from the Phishing Corpus [84]. The web server link is found in the message body:

http://61.143.38.56/www.paypal.com/page/update/

```
From: "Pay Pal" <do-not-reply@paypaI.com>
To: undisclosed-recipients: ;
Subject: New email address added to your account
Date: Thu, 16 Feb 2006 05:46:41 +0200

You have added steve85@aol.com as a new email address for
your account.

If you did not authorize this change or if you need assistance
with your account, please copy and paste the link in to your
internet browser:

http://61.143.38.56/www.paypal.com/page/update/
```

Figure 52 Phishing E-mail Example

Phishers often use similar behavior to drive traffic to their sites. In many instances, phishers use similar links for many phishing messages; e.g. re-using the same web server (registered domain), or the same path generated by a commonly used phishing kit (as phishing kits may simply contain zip archives allowing for quick installation of content for a phishing web site on a compromised server). The web server links in a message can be broken down into "n-grams", substrings of varying lengths. For example, possible substrings of interest for the message in the Figure 52 Phishing E-mail Example include "paypal" and "update". In order to discover which substrings should be used for measuring distance between messages, all substrings are derived for a "labeled" training corpus. We use all substrings where the Gini impurity measure of the phishing and non-phishing classes for the substring is less than ¼. The Jaccard distance between messages *i* and *j* is then computed as:

Equation 6.2 Jaccard Distance Between Messages

$$\text{distance}(i, j) = 1 - \frac{|substrings(i) \cap substrings(j)|}{|substrings(i) \cup substrings(j)|}$$

This is simply the complement of the proportion of URL substrings found in both messages.

6.3 Experimental Design

The Phishing Corpus [84] and the 2003 Spam Assassin Archive [85] are used for our experiments. The Phishing Corpus contains 4559 phishing messages, while the 2003 Spam Assassin archive contains 3900 "easy ham" messages, and 250 "hard ham" messages. These "ham" messages are messages that are not spam nor phishing messages. Our experiments focus on using messages containing URLs and little text content.

While some of the messages in the Phishing Corpus have relatively little content, such as the message shown in the Figure 52 Phishing E-mail Example, our experiments involve simulation of spear phishing with reduced content messages by removing text from the message. Tokens are randomly selected for removal. Somewhat surprisingly, the average number of tokens per message, as delimited by spaces and punctuation, is 517 tokens for phishing messages from the Phishing Corpus and 354 tokens for "ham" messages from the Spam Assassin archive. This is probably because phishers observed in the Phishing Corpus felt the need to put in extra content to make their message appear legitimate. In a spear phishing scenario, where a "friend" (or trusted contact) has sent a link simply saying "Check this out", this will not be the case. In order to simulate reduced content messages, we report results with 5%, 10%, 30%, 50%, and 100% of the

original content for the LDA approach. Furthermore, we use two different approaches to selecting content for removal. In the first approach, we use random selection of tokens for removal (RND). For the second approach, we use adversarial selection of tokens for removal (ADV). For adversarial selection, the probability of removing a token is proportional to the probability of a classifier identifying a message as a phishing message given the presence of a token within a set of "test" messages. This probability can be evaluated by an adversary by sending test messages and checking results, either via a web "beacon" (such as a web-based image) in the message for private e-mail systems or via "test" accounts for public e-mail systems (such as Gmail). The Spectral Clustering approach only relies on the URL links within the body of the message. URL links are not removed for either the LDA or Spectral Clustering approaches.

We provide both LDA and Spectral Clustering results for both the original message sets, and the modified message sets where a randomly selected subset of tokens is removed. A randomly selected subset of 500 messages containing URLs is selected from the combined corpus of 8709 messages, then k-fold ($k = 10$) cross validation is used to estimate the performance of both the LDA and the Spectral Clustering approaches.

For the LDA approach, the topic and term distributions discovered from the training corpus are applied to the test corpus. The number of topics in the training corpus is chosen as the number that minimizes the perplexity [78]. Lower perplexity scores mean a higher predicted likelihood for terms found in the corpus. A Random Forest is constructed for each training set, then evaluated on each test fold. For the Spectral Clustering approach, both the training and test sets are combined to derive the spectral

representation. The spectral representation, cluster membership, and silhouette values for each training set are used to build a Random Forest classifier, which is then evaluated on each test fold. For both the LDA approach and the Spectral Clustering approach, the Random Forests had 500 trees and the number of features considered for each split node was the ceiling of the square root of the number of features used for training.

E-mail classification for the Spectral Clustering approach uses transduction, rather than induction, for classification. Transduction focuses on reasoning from observed training examples to classify specific (observed) test examples, rather than generalizing to unobserved test examples (induction) [86] [87]. This is a form of Semi-Supervised Learning (SSL), as both the labeled training examples and the unlabeled test examples are analyzed together (via spectral decomposition) for classification. Batch processing of incoming messages can be performed frequently, including both messages observed earlier and newly received messages. For example, messages from the last hour can be clustered every 10 minutes to classify newly received messages. Message classification is performed as follows.

The first step in processing messages for the LDA approach is to extract tokens. Tokens are extracted by converting the text of the message subject and body to lower case; partitioning the text into tokens by breaking on spaces, tabs, and punctuation ('.', '?', '!'); and removing common stop words and tokens that occur in less than 1% of the messages. The Topic Modeling Toolkit (TMT) was then used to model the probability of each topic in the training messages; and the topic probabilities for each message were then derived for the test set.

The first step in processing messages for the Spectral Clustering approach is to extract URLs. The set of all possible substrings of length 1 through 10 is then considered for the URLs found in a message. Substrings which do not occur within at least 1% of all messages are discarded, as are substrings where the Gini impurity measure of the phishing and non-phishing message classes in the training set is greater than or equal to ¼.

6.4 Performance Evaluation

Performance is evaluated based on 5 classification metrics:

1. Area Under the Receiver Operating Characteristic (ROC) Curve (AUC): the probability that a randomly selected message from the phishing class will be viewed as more likely to be a phishing message than a randomly selected member of the non-phishing class
2. Accuracy: the probability that the predicted class for a randomly selected message is the actual class for that message
3. Precision: the probability that a predicted phishing message is actually a phishing message
4. Recall: the probability that an actual phishing message is predicted to be a phishing message
5. F Measure: the harmonic mean of Precision and Recall

Table 26 Results for Spectral Clustering Evaluation contains the results of our experiments.

Table 26 Results for Spectral Clustering Evaluation

Method	AUC	Accuracy	Precision	Recall	F Measure
SC	99.8%	97.6%	97.2%	98.6%	97.8%
LDA: 100%	99.4%	96.6%	97.1%	96.8%	96.9%
LDA: RND 50%	99.1%	95.8%	96.4%	96.0%	96.2%
LDA: RND 30%	98.8%	94.0%	94.9%	94.2%	94.6%
LDA: RND 10%	94.6%	88.2%	88.9%	89.9%	89.4%
LDA: RND 5%	90.0%	81.2%	82.1%	84.5%	83.3%
LDA: ADV 10%	79.6%	63.4%	79.7%	45.5%	57.9%
LDA: ADV 5%	71.6%	60.6%	73.5%	45.1%	55.9%

As seen in the Table 26 Results for Spectral Clustering Evaluation, the Latent Dirichlet Allocation (LDA) approach is found to be just as good as the Spectral Clustering (SC) approach when using 100% of the content, but SC does significantly better than LDA when the content of the message is reduced. Performance drops most significantly when content is removed proportionally to how well the content identifies a message as a phishing message.

The Figure 53 ROC Curves for Spectral Clustering Evaluation show the Receiver Operating Characteristic (ROC) curves for the experiments. The ROC curve [51] is formed by plotting the False Positive (FP) rate on the horizontal axis and the True Positive (TP) rate on the vertical axis as the classification threshold for identifying phish

is lowered from 1 to 0. The classification threshold is applied to the probability that a message is a phishing message: if the probability is above the classification threshold the message is treated as a phishing message, otherwise the message is treated as a non-phishing message.

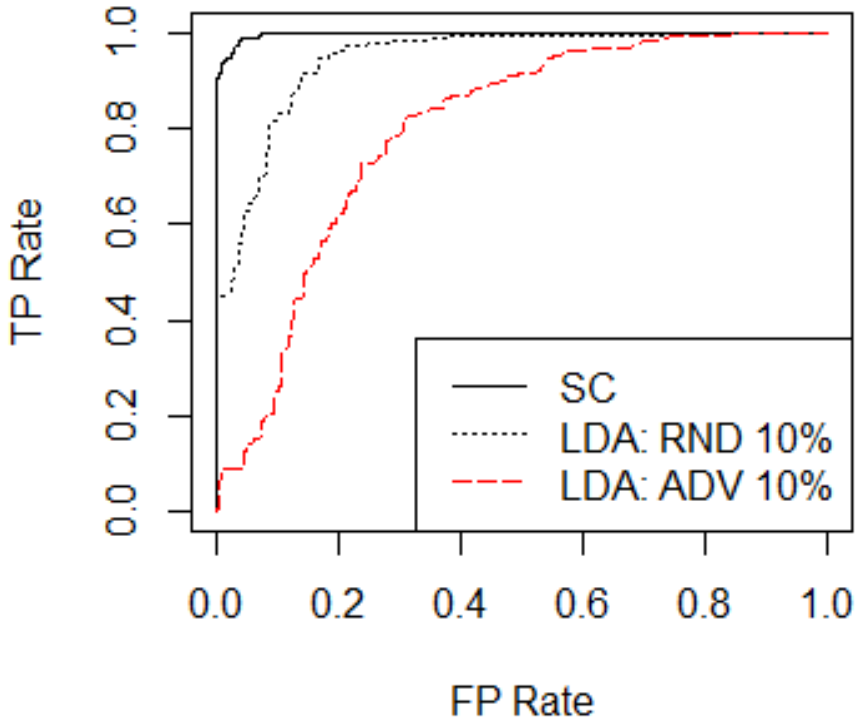

Figure 53 ROC Curves for Spectral Clustering Evaluation

As shown in the Figure 53 ROC Curves for Spectral Clustering Evaluation, the Spectral Clustering (SC) approach (solid black line) has the best performance. The true positive rate for a zero false positive rate drops below 50% when randomly selecting tokens for removal, and it drops below 10% when selecting tokens for removal proportional to their probability of identifying a message as a phishing message.

The optimal number of topics for the LDA approach was 45, as determined by average perplexity. An example of the top 3 terms for one of the LDA topics was "access", "limited", and "paypal" respectively. This makes sense as a topic that might often be found in phishing messages.

The optimal number of features for the Spectral Representation was 38, and the optimal number of clusters was 30. Examples of n-grams used for Spectral Clustering include "pay", "ebay", and "secur". These also make sense as substrings that might be found in phishing URLs that want to appear legitimate.

The most important features for the Spectral Clustering model, as measured by mean Gini decrease, are shown in Figure 54 Variable Importance for Spectral Clustering. Mean Gini decrease measures the average Gini decrease per tree for each feature. Assigned cluster membership and the silhouette value (a measure of how well an observation fits with the assigned cluster) are the two most important features used by the Random Forest for classification. The V# features are simply normalized eigenvector coordinates from the spectral representation, while "neighbor" is the identity of the next closest cluster (used for computing the silhouette value). It's interesting to note that the second eigenvector coordinate was considered to be much more important the first eigenvector coordinate, indicating that while the first coordinate captures more variance in link structure it has less utility for classification.

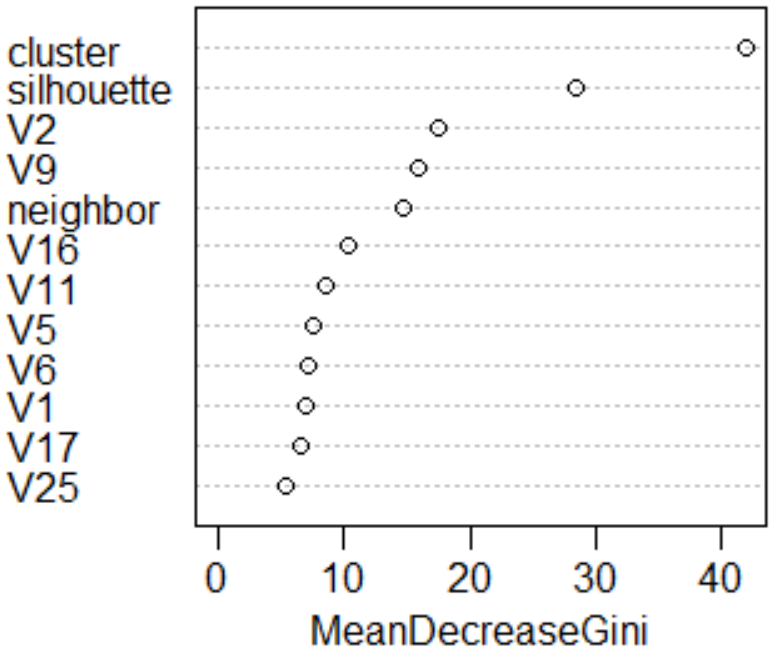

Figure 54 Variable Importance for Spectral Clustering

Multiple extensions are possible for this work. The measurement of similarity can be enhanced to include mail server address information found in the "Received" headers. This approach can detect "colluding providers" who act as phishing message carriers, though this would need to be done with care when using different message classes from different corpora (to avoid having message headers make separating the classes a trivial exercise). It may also be useful to determine whether messages are being sent person-to-person, or whether an organization or e-mail list is involved.

The Spectral Clustering approach can also be extended to support data streaming, where incoming messages are classified as they arrive. The challenge of spectral decomposition for large Laplacian matrices can also be addressed.

A similar Spectral Clustering approach can be evaluated for web site (rather than e-mail) classification too, as this might be useful for a browser. The web sites may contain limited content with hyperlinks to legitimate web sites.

Phishing is an attempt to steal a user's identity. This is typically accomplished by sending an e-mail message to a user, with a link directing the user to a web site used to collect account information. Phishing detection systems typically rely on content filtering techniques, such as Latent Dirichlet Allocation (LDA), to identify phishing messages. In the case of spear phishing, however, this may be ineffective; because messages from a trusted source may contain little content. In order to identify this type of spear phishing behavior, Spectral Clustering of messages was proposed, based on analysis of links between the URLs for web sites found in the message contents. Cluster membership is then used to construct a classifier for phishing. Data from the Phishing E-mail Corpus and the Spam Assassin E-mail Corpus were used to evaluate this approach. Performance evaluation metrics included the area under the Receiver Operating Characteristic (ROC) curve, as well as accuracy, precision, recall, and the F measure. When 100% of the message content was present the Spectral Clustering approach was just as good as the LDA approach; but for experiments where the content was randomly reduced, the Spectral Clustering approach was significantly better. For the type of spear phishing studied, performance of the proposed Spectral Clustering approach is found to provide significant improvements in all metrics evaluated, including 9.4% and 68.9% improvements in the F measure when the content of the messages is reduced to 10% of their original size using random and adversarial deletion, respectively. The lift in

performance is significantly larger when adversarial phishers choose which terms to remove based on whether a term is likely to increase the probability of a message being classified as phishing.

7. FRAUD DETECTION

Fraud is the use of deception to gain some benefit, often financial gain. Examples of fraud include insurance fraud, credit card fraud, telecommunications fraud, securities fraud, and accounting fraud. Costs for the affected companies are high, and these costs are passed on to their customers. Detection of fraudulent activity is thus critical to control these costs. Last but not least, in order to avoid detection, fraudsters often change their "signatures" (methods of operation). We propose here to address insurance fraud detection via the use of reputation features that characterize insurance claims and ensemble learning to compensate for varying data distributions. We replace each of the original features in the data set with 5 reputation features (RepF): 1) a count of the number of fraudulent claims with the same feature value in the previous 12 months, 2) a count of the number of months in the previous 12 months with a fraudulent claim with the same feature value, 3) a count of the number of legitimate claims with the same feature value in the previous 12 months, 4) a count of the number of months in the previous 12 months with a legitimate claim with the same feature value, and 5) the proportion of claims with the same feature value which are fraudulent in the previous 12 months. Furthermore we use two one-class Support Vector Machines (SVMs) to measure the similarity of the derived reputation feature vector to recently observed fraudulent claims and recently observed legitimate claims. The combined reputation and

similarity features are then used to train a Random Forest classifier for new insurance claims. A publicly available auto insurance fraud data set is used to evaluate our approach. Cost savings, the difference in cost for predicting all new insurance claims as non-fraudulent and predicting fraud based on a trained data mining model, is used as our primary evaluation metric. Our approach shows a 13.5% increase in cost savings compared to previously published state of the art results for the auto insurance fraud data set.

The novelty of our approach to fraud detection is the use of reputation features and one-class SVM similarity features for fraud detection. Reputation features have been used in [88] to analyze the previous behavior of Wikipedia editors for vandalism detection. For fraud detection, reputation features are used to characterize how often feature values from a claim have been associated with fraud in the past. Similarity features, derived from one-class SVMs, are then used to extend reputation from individual features to the joint distribution of features for a claim. Finally, a cost-sensitive Random Forest classification model is constructed to classify new claims based on reputation and similarity features.

Unfortunately there is not much publicly available fraud detection data available for research. Corporate victims of fraud are often reluctant to admit that they have been the victims of fraud, and transactional data is often sensitive (e.g. containing personally identifiable account information). The auto insurance fraud data set used in this study is the only publicly available fraud detection data set that we are aware of.

7.1 Background

The presence of an imbalanced class distribution is a common characteristic for fraud detection applications [19] [89], because fraudulent transactions occur much less frequently than non-fraudulent transactions. For some domains, fraud may occur 10 or more times less frequently than non-fraudulent transactions. Because there are relatively few fraudulent transactions compared to non-fraudulent transactions, larger data sets are required to learn to confidently distinguish fraudulent transactions from non-fraudulent transactions.

To make matters worse, fraudulent transactions often look like non-fraudulent transactions because the fraudsters want to avoid detection; i.e. the fraudulent and non-fraudulent classes overlap. Because fraudulent transactions look like non-fraudulent transactions (the classes overlap), standard pattern recognition "learning" algorithms will make fewer errors by simply declaring all transactions to be non-fraudulent. The resulting classification model is known as a "majority" classifier, because it simply declares all transactions to belong to the majority class (non-fraudulent transactions). Additionally, fraudsters may adapt their observed behavior in response to detection [19]. This leads to an adversarial game in which detection advocates must somehow adapt to changes made by fraudsters, which in turn leads fraudsters to adapt to changes made by detection advocates. The resulting changes in the data distribution are known as concept drift [90].

Consider two overlapping distributions: a bivariate Gaussian distribution (with 2 independent features) centered at (2,2) with a standard deviation of 0.5, and a second

bivariate Gaussian distribution (with 2 independent features) centered at (0,0) with a standard deviation of 2. Suppose that the prior probability for the class centered at (2,2) is 1% and the prior probability for the class centered at (0,0) is 99%. In this situation, a majority classifier would be the optimal Bayes classifier [91] if the misclassification costs are equal! Bayesian risk for predicting class α_i for observation **x** is computed as:

Equation 7.1 Bayesian Risk

$$R(\alpha_i \mid \mathbf{x}) = \sum_{j=1}^{C} \lambda(\alpha_i \mid \omega_j) P(\omega_j \mid \mathbf{x})$$

where $\lambda(\alpha_i \mid \omega_j)$ is the loss (misclassification cost) associated with predicting class α_i when the actual class is ω_j and

$$P(\omega_j \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid \omega_j) P(\omega_j)}{\sum_{k=1}^{C} p(\mathbf{x} \mid \omega_k) P(\omega_k)}$$

where $P(\omega_j \mid \mathbf{x})$ is the posterior probability of observation $\mathbf{x}$ belonging to class ω_j, $p(\mathbf{x} \mid \omega_j)$ is the likelihood (density estimate) of observing $\mathbf{x}$ within class ω_j, $P(\omega_j)$ is the prior probability of observing class ω_j, C is the number of classes (2 for fraud detection), and $\sum_{k=1}^{C} p(\mathbf{x} \mid \omega_k) P(\omega_k)$ is the evidence for observation $\mathbf{x}$.

Further suppose that the minority class represents fraud. The principal costs for fraud are the cost of investigations and the cost of paying claims. If an investigation costs $100 and a claim costs $5000, then the decision boundary for the optimal Bayes classifier is shown by the solid line in Figure 55 Optimal Bayes Decision Boundary.

The optimal Bayes classifier predicts an observation to be a member of the "positive" class iff ...

$$\left(x_1 - \frac{32}{15}\right)^2 + \left(x_2 - \frac{32}{15}\right)^2 < \left(\frac{2}{15}\sqrt{30 \ln \frac{784}{99} + 32}\right)^2$$

95% of the fraudulent class distribution lies in the small dotted circle, while 95% of the non-fraudulent class distribution lies in the large dotted circle.

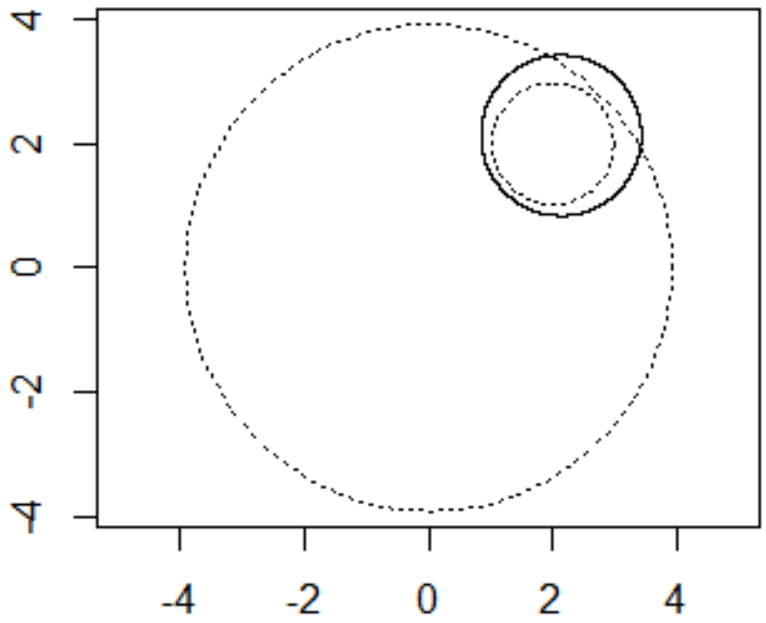

Figure 55 Optimal Bayes Decision Boundary

As shown in the Table 27 Optimal Bayes Classification Errors, this classifier would misclassify 4.4% of the fraudulent class distribution as non-fraudulent and 6.8% of the non-fraudulent class distribution as fraudulent; but overall costs would be minimized,

as the expected cost of a claim would be 100 * .956 * .01 + 5000 * .044 * .01 + 5100 * .068 * .99 + 5000 * .932 * .99 = $4,960.

Table 27 Optimal Bayes Classification Errors

		Predict	
		Fraud	Not Fraud
Actual	Fraud	95.6%	4.4%
	Not Fraud	6.8%	93.2%

Strategies for overcoming the tendency to produce a simple "majority" classifier include sampling or cost sensitive learning [89] [92]. For sampling strategies, the training examples are "stratified" (partitioned) into two groups: fraudulent training examples and non-fraudulent training examples. Sampling from the training set can be performed "with" or "without" replacement. In sampling "with" replacement, each training example can be selected more than once, while in sampling "without" replacement, each training example can be selected at most once. In order to balance the fraudulent and non-fraudulent training examples, the majority class can be under-sampled or the minority class can be over-sampled. Under-sampling involves selecting a subset of the non-fraudulent training examples, while over-sampling involves selecting a superset of the fraudulent training examples. Selecting a superset of the fraudulent training examples can be performed by sampling with replacement, or even synthesizing new fraudulent training examples similar to known fraudulent training examples [93].

In cost sensitive learning, the training examples are weighted to reflect different misclassification costs. Fraudulent training examples are given a larger weight than the non-fraudulent training examples, reflecting the notion that misclassifying a fraudulent example as non-fraudulent has a higher cost than misclassifying a non-fraudulent training example as fraudulent. This can be viewed as an alternative form of a sampling strategy, where the use of larger weights is a form of over-sampling and the use of smaller weights is a form of under-sampling. Strategies for handling imbalanced class problems can be used with any pattern recognition algorithm, including decision trees, rules, neural networks, Support Vector Machines, and others. This includes the use of bagging and boosting with the MetaCost framework [92].

7.2 Proposed Method

The training process for the proposed approach to fraud detection is illustrated in the Figure 56 Reputation Features Training Process. As shown, our first step is to compute reputation features.

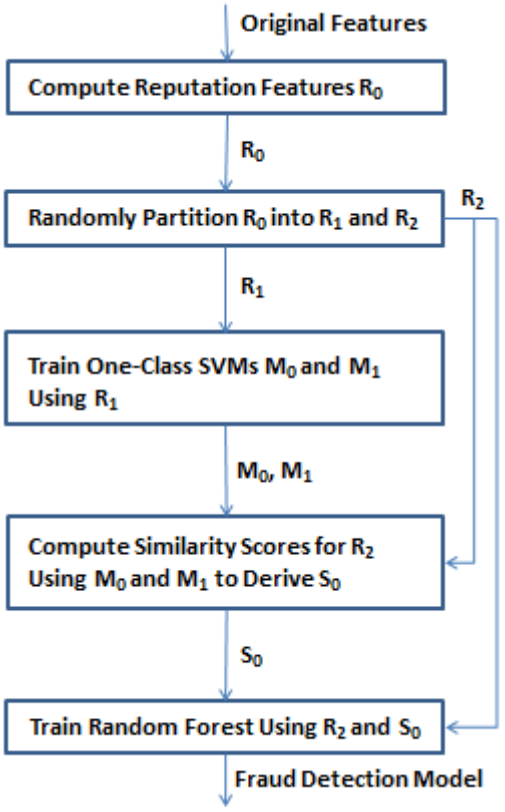

Figure 56 Reputation Features Training Process

We propose replacing each of the original features in the data set with 5 reputation features:

1. *Fraud Count*: a count of the number of fraudulent claims with the same feature value in the previous 12 months,
2. *Fraud Months*: a count of the number of months in the previous 12 months with a fraudulent claim with the same feature value,
3. *Legitimate Count*: a count of the number of legitimate claims with the same feature value in the previous 12 months,

4. *Legitimate Months*: a count of the number of months in the previous 12 months with a legitimate claim with the same feature value, and
5. *Fraud Rate*: the proportion of claims with the same feature value which are fraudulent in the previous 12 months.

These 5 values capture support and confidence for each feature value: how often a particular value is observed for a class (support) and what proportion of the time a particular value is associated with fraud (confidence). For the proportion feature we use a Wilson estimate [94] of the proportion to avoid the extremes of zero or one when we have not observed the value very often in previous months. The Wilson estimate is a weighted average of the observed proportion and one half:

Equation 7.2 Wilson Estimate of Proportion

$$\hat{p} = \frac{1*\left(\frac{FraudCount}{FraudCount + LegitimateCount}\right) + \frac{1.96^2}{TotalCount} * \left(\frac{1}{2}\right)}{1 + \frac{1.96^2}{TotalCount}}$$

A training set is then randomly partitioned into two equal size subsets. The first subset is used to derive two one-class SVMs, while the second subset is used to construct a Random Forest classifier using the reputation and one-class SVM similarity features.

The two one-class Support Vector Machines (SVMs) measure the similarity of the derived feature vector to previously observed fraudulent claims and previously observed legitimate claims. One class SVMs [35] are used to determine whether an observation is consistent with previous observations for some class of interest.

The radial basis function was used as the kernel function for our one class SVM models:

$$K\left(\mathbf{x}_i, \mathbf{x}_j\right) = \exp\left(-\frac{\left\|x_i - x_j\right\|^2}{\sigma^2}\right)$$

where σ is the mean of the 10th, 50th, and 90th percentile of Euclidean distance values for a random sample of *n/2* pairs of observations [95]. The hyper-parameter ν for the one class SVM places an upper bound on the proportion of the training data that can be declared to be outliers and a lower bound on the proportion of the training set to be used as support vectors. The value of ν was chosen to be 0.05 for our experiments.

Once the α_i coefficients have been found, the distance of a new observation from the class boundary defined by the one-class SVM can be computed as:

$$\sum_{i=1}^{n}\left(\alpha_i K\left(\mathbf{x}, \mathbf{x}_i\right)\right) - \rho$$

where ρ is chosen as the offset that yields at least $1 - \upsilon n$ positive values for observations of the training set. The output of the one-class SVM is a measure of how well a new observation fits with the observed training distribution. The larger the similarity feature value, the more likely the observation belongs to the distribution characterized by the training data. The probability of membership can be estimated by comparing the distance for a new observation to the distance values computed for the training data.

To generate 2 new features for input to a classification model, two one-class SVMs are constructed using the first subset of training data. The first one-class SVM is constructed from fraudulent transactions in the first subset of training data, while the

second one-class SVM is constructed from non-fraudulent transactions in the first subset of training data. Unlike the other reputation features, the one-class SVM similarity features consider the joint distribution of feature values when evaluating feature vectors.

The derived feature vectors for the second subset of training data, including the two one-class SVM similarity features, are used to construct a Random Forest classifier [50].

To support cost sensitive learning, we used a balanced stratified sampling approach [96] to generate bootstrap samples for training the classifier. For training each tree, a bootstrap sample is drawn from the minority class and a sample of the same size is drawn (with replacement) from the majority class. This effectively under-samples the majority class.

To classify new feature vectors, the reputation features and two one-class SVM similarity features are derived, then each tree in the Random Forest classification model casts its vote for a class label: fraud or not fraud. The proportion of votes for the fraud class is the probability that a randomly selected tree would classify the feature vector as belonging to the fraud class. This is interpreted as the probability of a feature vector belonging to the fraud class.

7.3 Experimental Design

The auto insurance data set [97] has been used to demonstrate fraud detection capabilities [98] [9]. As this is the only publicly available fraud data set, we use it for our experiments as well. It consists of 3 years of auto insurance claims: 1994, 1995, and

1996. The Table 28 Fraud Rates for Auto Insurance Data describe the distribution of fraudulent and non-fraudulent claims by year.

Table 28 Fraud Rates for Auto Insurance Data

Year	Fraud	Not Fraud	Fraud Rate
1994	409	5,733	6.7%
1995	301	4,894	5.8%
1996	213	3,870	5.2%
All	923	14,497	6.0%

The proportion of overall claims that are fraudulent is only 6%, so only 1 in 17 claims are fraudulent. Table 29 Original Features for Auto Insurance Data lists the features of the data. As shown, only two of the features were not used for prediction. The Year attribute obviously does not generalize to future data; and since we assume that policies associated with known fraudulent activity are terminated, we ignore the Policy Number attribute as well.

Table 29 Original Features for Auto Insurance Data

Month	Fault	Past Number of Claims
Week of Month	Policy Type	Age of Vehicle
Day of Week	Vehicle Category	Age of Policy Holder
Make	Vehicle Price	Police Report Filed
Accident Area	Fraud Found	Witness Present
Day of Week Claimed	~~Policy Number~~	Agent Type
Month Claimed	Rep Number	Number of Suppliments
Week of Month Claimed	Deductible	Address Change Claim
Sex	Driver Rating	Number of Cars
Marital Status	Days Policy Accident	~~Year~~
Age	Days Policy Claim	Base Policy

Values in the data set have been pre-discretized (probably for anonymization); e.g. the distribution of Vehicle Price appears in Table 30 Vehicle Price Distribution.

Table 30 Vehicle Price Distribution

Value	Frequency
Less than 20,000	1,096
20,000 to 29,000	8,079
30,000 to 39,000	3,533
40,000 to 59,000	461
60,000 to 69,000	87
More than 69,000	2,164

We constructed Random Forest classification models for both the original features and the reputation features, as described in the previous section. To be consistent with previously reported results, claims from 1994 and 1995 were used as training data, and claims from 1996 were used as testing data. The primary evaluation measure is cost savings, as this was reported in earlier publications and it directly relates to the core goal for a fraud detection system: cost reduction.

Given classification results, as shown in the Table 10 Confusion Matrix, costs can be computed as follows:

Equation 7.3 Total Cost of a Fraud Detection Model

$$TotalCost = InvestigationCost * TP + (InvestigationCost + ClaimCost) * FP + ClaimCost * (TN + FN)$$

In [9], the average InvestigationCost was given as $203 and the average ClaimCost was given as $2,640. We use these values as well for consistency. We ran 10 trials (based on random sampling) for both the Original Features (OrigF) approach and the Reputation Features (RepF) approach.

Using the Original Features (OrigF), we constructed 10 Random Forests from the 11,337 original feature vectors from 1994 and 1995. Each of these 10 Random Forests was evaluated on the 4,083 original feature vectors from 1996.

Using the Reputation Features (RepF), we partitioned the 5,195 reputation feature vectors from 1995 into two subsets (as we used 12 months of history to construct reputation features). For 10 iterations, the first subset was used to construct our one-class SVMs and the second subset was used to construct a Random Forest classifier. Each of the 10 Random Forests was then evaluated on the 4,083 reputation feature vectors from 1996.

Balanced stratified random sampling was used for constructing Random Forests for both the original features and the reputation features. A total of 2,000 trees were constructed for each Random Forest model, with the ceiling of the square root of the number of input features used as the number of randomly selected features to consider for each decision node. For both Original Features (OrigF) and Reputation Features (RepF) the Out Of Bag (OOB) estimate of error from the training data [50] was used to select the classification threshold which minimizes cost.

7.4 Performance Evaluation

Cost savings is used as our primary metric of interest. In [9], cost savings was recorded as the difference between the cost of paying all claims and the cost of using a fraud detection model (Equation 11). In addition to cost savings, we also report the following metrics:

1. Area Under the Receiver Operating Characteristic (ROC) Curve (AUC): the probability that a randomly selected claim from the fraud class will be viewed as more likely to be a fraudulent claim than a randomly selected claim from the not-fraud class
2. Precision: the probability that a predicted fraudulent claim is actually a fraudulent claim
3. Recall: the probability that an actual fraudulent claim is predicted to be a fraudulent claim
4. F Measure: the harmonic mean of Precision and Recall

Table 31 Performance for the Reputation Features Evaluation shows evaluation metrics for our experiments. The values for the Reputation Features approach are marked as RepF, while the values for the Original Features approach are marked as OrigF. Standard Error values are listed (in parentheses) to assess statistical significance.

Table 31 Performance for the Reputation Features Evaluation

	Reputation Features	Original Features
Cost Savings	$189,651 ($2,665)	$165,808 ($748)
AUC	82.0% (0.1%)	73.8% (< 0.1%)
Precision	13.3% (0.1%)	11.2% (< 0.1%)
Recall	80.3% (1.1%)	94.2% (0.1%)
F Measure	22.8% (0.1%)	20.0% (< 0.1%)

The Table 32 Confusion Matrix for the Reputation Features Approach shows the average confusion matrix for the Reputation Features approach.

Table 32 Confusion Matrix for the Reputation Features Approach

		Predicted	
		Fraud	Not Fraud
Actual	Fraud	171.0	42.0
	Not Fraud	1,118.6	2,751.4

The Table 33 Confusion Matrix for the Original Features Approach shows the average confusion matrix for the Original Features approach.

Table 33 Confusion Matrix for the Original Features Approach

		Predicted	
		Fraud	Not Fraud
Actual	Fraud	200.6	12.4
	Not Fraud	1,591.4	2,278.6

The Figure 57 ROC Curves for Reputation Features Evaluation compare the Receiver Operating Characteristic (ROC) curves for the two approaches to fraud detection.

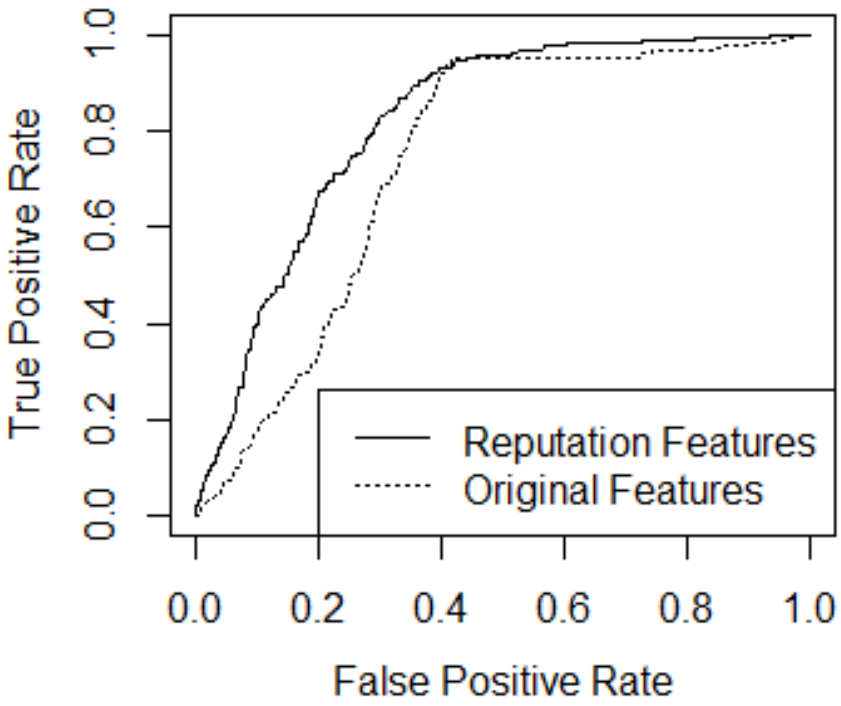

Figure 57 ROC Curves for Reputation Features Evaluation

Figure 58 Variable Importance for Reputation Features identifies the most important classification features for the Reputation Features approach. Both the one-class SVM similarity feature for the Fraud class and the Not-Fraud (Legitimate) class are identified as important features.

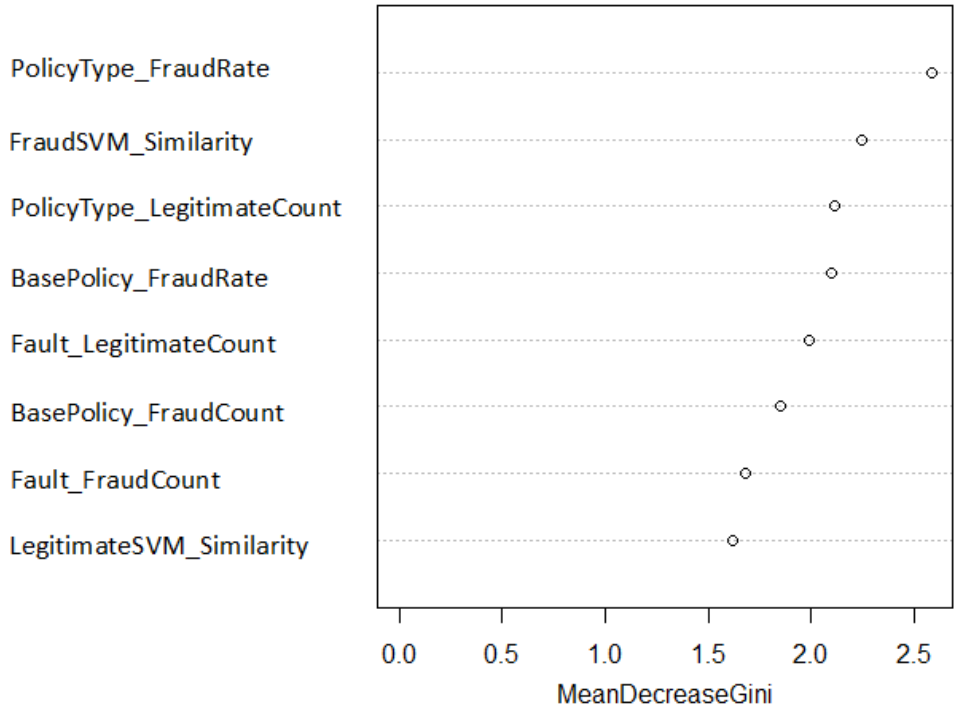

Figure 58 Variable Importance for Reputation Features

As shown in Table 31 Performance for the Reputation Features Evaluation, the Random Forest classifier constructed from the Original Features is competitive with previously reported state-of-the-art results. The previously reported state-of-the-art result for cost savings was $167,069, while the upper bound of the 95% confidence interval for the Original Features approach shown in Table 31 Performance for the Reputation

Features Evaluation is $165,808 + 1.96 * 748 = $167,274. The cost savings for the Reputation Features approach is 13.5% higher than the previously reported state-of-the-art results:

($189,651 - $167,069) / $167,069 = 13.5%

It's interesting to note that the operating threshold for the Original Features approach occurs where the two ROC curves meet; but the operating threshold for the Reputation Features approach occurs in the region where the False Positive rate is 10% lower. Though the True Positive rate (recall) is lower for the Reputation Features approach, the overall cost is significantly reduced.

The use of deception for financial gain is a commonly encountered form of fraud. Costs for the affected companies are high, and these costs are passed on to their customers. Detection of fraudulent activity is thus critical to control these costs. We proposed to address insurance fraud detection via the use of reputation and similarity features that characterize insurance claims and ensemble learning to compensate for changes in the underlying data distribution. A publicly available auto insurance fraud data set was used to evaluate our approach. Our approach showed a 13.5% increase in cost savings compared to previously published state of the art results for the auto insurance fraud data set. Though an auto insurance fraud data set was used for this demonstration, reputation features could easily be applied to other fraud detection domains, including health care insurance fraud, credit card fraud, securities fraud, and accounting fraud. This approach could also be useful for other applications, such as credit risk classification [99] or computer network intrusion detection [100].

Future extensions include investigation into the use of alternative reputation history lengths. For example, we will explore the use of reputation features based on the most recent 3, 6 and 9 month intervals (in addition to the existing 12 month interval). We could also investigate the utility of updating the one-class SVMs on a monthly basis, and synthesizing data to show robustness against adversarial changes to the underlying data distribution for the fraud class.

8. DISCUSSION

Possible extensions of this research includes scaling the proposed solutions to larger data sets, so called "big data." Obtaining annotated data will still be expensive and error prone; and adversaries will still attempt to avoid detection. The methods proposed here are easily ported to large scale map/reduce platforms, such as Hadoop. The observations are distributed (mapped) to processors based on key values, then reduced via summarization and/or learning. Parallel implementations already exist for clustering and Random Forests in Mahout, though modification for both may be needed to support dense data vectors or imbalanced classes (as in fraud detection). Furthermore, the following algorithms can be easily parallelized for the Hadoop environment:

- Gradient Boosting Machines, including the logistic loss function, the adaptive boosting (adaBoost) loss function, and the Huberized hinge loss functions
- Sequential Minimal Optimization
- Eigenvalue decomposition (for spectral clustering)

9. CONCLUSIONS

Spam, phishing, and fraud detection are security applications that impact most people. Spam causes users to waste time reviewing unwanted messages and prevents users from receiving relevant messages. Phishing leads to identity theft. Fraud causes companies to lose money and pass along costs to consumers. Challenges for building spam, phishing, and fraud detection models include difficulty in obtaining annotated data, increased computational complexity for robust detection methods, annotation errors, and changes in the underlying data distribution. In this research, clustering, active learning, reputation features, random projections, and semi-supervised learning are used to address these issues. Clustering and active learning are used together to make efficient use of annotated data, yielding state of the art spam detection performance using only 10% of the annotated data employed by previously published methods. Social Network Analysis (SNA) reputation features are introduced to evaluate paths from sender to receiver, increasing the detection rate by 70% for state of the art spam detection. Random projections are used with boosting, achieving state of the art spam detection with a 75% reduction in computational cost for message classification. The Randomized Hough Transform – Support Vector Machine is used to update training set annotations, increasing the F measure by 9.3% comparing to a state of the art method for handling adversarial noise. Spectral clustering of URL n-grams and transductive Semi-Supervised

Learning (SSL) increase the detection rate by 100% for state of the art phishing detection under adversarial modification of message text. Reputation and similarity features are used to enhance the ability to withstand changes in underlying data distributions, producing a 13.5% increase in cost savings for state of the art fraud detection. Future research possibilities include the application of these methods to identify deception in social media channels.

REFERENCES

[1] D. DeBarr and H. Wechsler, "Spam Detection Using Clustering, Random Forests, and Active Learning," in *Proceedings of the 6th Conference on E-mail and Anti-Spam (CEAS)*, Mountain View, CA, 2009.

[2] D. DeBarr and H. Wechsler, "Using Social Network Analysis for Spam Detection," in *Proceedings of the 3rd International Conference on Social Computing, Behavioral Modeling, and Prediction (SBP)*, Bethesda, MD, 2010.

[3] D. DeBarr and H. Wechsler, "Spam Detection Using Random Boost," *Pattern Recognition Letters*, vol. 33, no. 10, pp. 1237-1244, 2012.

[4] D. DeBarr, H. Sun and H. Wechsler, "Adversarial Spam Detection Using the Randomized Hough Transform - Support Vector Machine," in *Institute for Electrical and Electronics Engineers (IEEE) International Conference on Machine Learning and Applications (ICMLA)*, Miami, FL, 2013.

[5] D. DeBarr, V. Ramanathan and H. Wechsler, "Phishing Detection Using Traffic Behavior, Spectral Clustering, and Random Forests," in *Proceedings of the Institute for Electrical and Electronics Engineers (IEEE) Conference on Intelligence and Security Informatics (ISI)*, Seattle, WA, 2013.

[6] D. DeBarr and H. Wechsler, "Fraud Detection Using Reputation Features, Support Vector Machines, and Random Forests," in *Proceedings of the 9th International Conference on Data Mining*, Las Vegas, NV, 2013.

[7] G. Cormack, "Text Retrieval Conference (TREC) 2007 Spam Track Overview," in *Proceedings of TREC 2007: The 16th Text Retrieval Conference*, Gaithersburg, MD, 2007.

[8] V. Ramanathan and H. Wechsler, "Phishing Detection and Impersonated Entity Discovery Using Conditional Random Field and Latent Dirichlet Allocation," *Computers & Security*, vol. 34, no. 1, pp. 123-139, 2013.

[9] C. Phua, D. Alahakoon and V. Lee, "Minority Report in Fraud Detection: Classification of Skewed Data," *Special Interest Group on Knowledge Discovery from Data (SIGKDD) Explorations,* vol. 6, no. 1, pp. 50-59, 2004.

[10] Symantec MessageLabs, "August 2010 Intelligence Report," Symantec MessageLabs, Mountain View, CA, 2010.

[11] J. M. Rao and D. M. Reiley, "The Economics of Spam," *Journal of Economic Perspectives,* vol. 26, no. 3, pp. 87-110, 2012.

[12] J. B. Postel, *Request For Comments (RFC) 821: Simple Mail Transfer Protocol (SMTP),* Marina del Rey, CA: Internet Engineering Task Force (IETF), 1982.

[13] D. H. Crocker, *Request For Comments (RFC) 822: Standard for the Format of Advanced Research Projects Agency (ARPA) Internet Text Messages,* Newark, DE: Internet Engineering Task Force (IETF), 1982.

[14] Alias-I, "LingPipe 4.1.0," 1 October 2008. [Online]. Available: http://alias-i.com/lingpipe. [Accessed 20 September 2013].

[15] M. F. Porter, "An Algorithm for Suffix Stripping," *Program,* vol. 14, no. 3, pp. 130-137, 1980.

[16] Federal Bureau of Investigation (FBI), "Financial Crimes Report to the Public: 2010-2011," 2012. [Online]. Available: http://www.fbi.gov/stats-services/publications/financial-crimes-report-2010-2011. [Accessed 20 September 2013].

[17] Association of Certified Fraud Examiners, "2012 Report to the Nations," 2012. [Online]. Available: http://www.acfe.com/uploadedFiles/ACFE_Website/Content/rttn/2012-report-to-nations.pdf. [Accessed 20 September 2013].

[18] D. DeBarr and Z. Eyler-Walker, "Closing the Gap: Automated Screening of Tax Returns to Identify Egregious Tax Shelters," *Special Interest Group on Knowledge Discovery and Data Mining (SIGKDD) Explorations,* vol. 8, no. 1, pp. 11-16, 2006.

[19] R. J. Bolton and D. J. Hand, "Statistical Fraud Detection: A Review," *Statistical Science,* vol. 17, no. 3, pp. 235-255, 2002.

[20] J. Li, K.-Y. Huang, J. Jin and J. Shi, "A Survey on Statistical Methods for Health Care Fraud Detection," *Health Care Management Science,* vol. 11, no. 3, pp. 275-287, 2008.

[21] S. Viaene, R. A. Derrig and G. Dedene, "A Case Study of Applying Boosting Naive Bayes to Claim Fraud Diagnosis," *Institute for Electrical and Electronics Engineers (IEEE) Transactions on Knowledge and Data Engineering,* vol. 16, no. 5, pp. 612-620, 2004.

[22] P. K. Chan and S. J. Stolfo, "Toward Scalable Learning with Non-Uniform Class and Cost Distributions: A Case Study in Credit Card Fraud Detection," in *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining (KDD)*, New York, NY, 1998.

[23] R. A. Becker, C. Volinsky and A. R. Wilks, "Fraud Detection in Telecommunications: History and Lessons Learned," *Technometrics,* vol. 52, no. 1, pp. 20-33, 2010.

[24] T. Fawcett and F. Provost, "Adaptive Fraud Detection," *Data Mining and Knowledge Discovery,* vol. 1, no. 3, pp. 291-316, 1997.

[25] T. Hastie, R. Tibshirani and J. Friedman, "Model Assessment and Selection," in *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, New York, Springer, 2011, pp. 219-260.

[26] V. Cherkassky and F. M. Mulier, "Statistical Learning Theory," in *Learning from Data: Concepts, Theory, and Methods*, Hoboken, NJ, John Wiley & Sons Inc, 2007, pp. 99-150.

[27] L. Breiman, J. H. Friedman, R. A. Olshen and C. J. Stone, Classification And Regression Trees (CART), Boca Raton, FL: Chapman & Hall/CRC, 1984.

[28] J. C. Platt, "Fast Training of Support Vector Machines Using Sequential Minimal Optimization," in *Advances in Kernel Methods: Support Vector Learning*, Cambridge, MA, The Massachusetts Institute of Technology (MIT) Press, 1998, pp. 185-208.

[29] C.-C. Chang and C.-J. Lin, "LIBSVM: A Library for Support Vector Machines," *Association for Computing Machinery (ACM) Transactions on Intelligent Systems and Technology,* vol. 2, no. 3, pp. 27:1-27:27, 2011.

[30] R.-E. Fan, P.-H. Chen and C.-J. Len, "Working Set Selection Using Second Order Information for Training Support Vector Machines," *Journal of Machine Learning Research (JMLR),* vol. 6, no. 12, pp. 1889-1918, 2005.

[31] J. Friedman, T. Hastie and R. Tibshirani, "Additive Logistic Regression: a Statistical View of Boosting," *Annals of Statistics,* vol. 28, no. 2, pp. 337-407, 2000.

[32] J. H. Friedman, "Stochastic Gradient Boosting," *Computational Statistics & Data Analysis,* vol. 38, no. 4, pp. 367-378, 2002.

[33] L. Breiman, "Bias, Variance, and ARCing Classifiers," University of California - Berkeley, Berkeley, CA, 1996.

[34] L. Breiman, "Bagging Predictors," *Machine Learning,* vol. 24, no. 2, pp. 123-140, 1996.

[35] B. Scholkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola and R. C. Williamson, "Estimating the Support of a High-Dimensional Distribution," Microsoft Research, Redmond, WA, 1999.

[36] X. Zhu, "Semi-Supervised Learning Literature Survey," University of Wisconsin - Madison, Madison, WI, 2008.

[37] O. Chapelle, B. Scholkopf and A. Zien, "Introduction to Semi-Supervised Learning," in *Semi-Supervised Learning*, Cambridge, MA, The Massachusetts Institute of Technology (MIT) Press, 2006, pp. 1-12.

[38] E. Riloff, J. Wiebe and T. Wilson, "Learning Subjective Nouns using Extraction Pattern Bootstrapping," in *Proceedings of the 7th Conference on Natural Language Learning (CoNLL)*, Edmonton, Canada, 2003.

[39] V. N. Vapnik, "Estimating the Values of Functions at Given Points," in *Statistical Learning Theory*, New York, John Wiley & Sons Inc, 1998, pp. 339-371.

[40] D. Lowd and C. Meek, "Adversarial Learning," in *Proceedings of the 11th International Conference on Knowledge Discovery and Data Mining (KDD)*, Chicago, IL, 2005.

[41] R. Kohavi and F. Provost, "Glossary of Terms," *Machine Learning,* vol. 30, no. 3, pp. 271-274, 1998.

[42] H. B. Mann and D. R. Whitney, "On a Test of Whether One of Two Random Variables is Stochastically Larger than the Other," *The Annals of Mathematical Statistics,* vol. 18, no. 1, pp. 50-60, 1947.

[43] G. V. Cormack, "Email Spam Filtering: a Systematic Review," *Foundations and Trends in Information Retrieval,* vol. 1, no. 4, pp. 335-455, 2007.

[44] J. Goodman and W.-t. Yih, "Online Discriminative Spam Filter Training," in *Proceedings of the 3rd Conference on Email and Anti Spam (CEAS)*, Mountain View, CA, 2006.

[45] D. Sculley and G. M. Wachman, "Relaxed Online SVMs in the Text REtrieval Conference (TREC) Spam Filtering Track," in *Proceedings of the 16th Text REtrieval Conference (TREC)*, Gaithersburg, MD, 2007.

[46] B. Settles, "Active Learning Literature Survey," University of Wisconsin-Madison Computer Sciences Technical Report 1648, Madison, WI, 2009.

[47] C. D. Manning, P. Raghavan and H. Schutze, "Scoring, Term Weighting, and the Vector Space Model," in *Introduction to Information Retrieval (IR)*, New York, NY, Cambridge University Press, 2008, pp. 100-123.

[48] H. T. Nguyen and A. Smeulders, "Active Learning Using Pre-Clustering," in *Proceedings of the 21st International Conference on Machine Learning (ICML)*, Banff, Alberta, Canada, 2004.

[49] L. Kaufman and P. J. Rousseeuw, "Partitioning Around Medoids (Program PAM)," in *Finding Groups in Data: an Introduction to Cluster Analysis*, Hoboken, NJ, John Wiley & Sons Inc, 2005, pp. 68-125.

[50] L. Breiman, "Random Forests," *Machine Learning,* vol. 45, no. 1, pp. 5-32, 2001.

[51] T. Fawcett, "An Introduction to Receiver Operating Characteristic (ROC) Analysis," *Pattern Recognition Letters,* vol. 27, no. 8, pp. 861-874, 2006.

[52] D. D. Lewis and W. A. Gale, "A Sequential Algorithm for Training Text Classifiers," in *Proceedings of the 17th Annual Association for Computing Machinery Special Interest Group on Information Retrieval Conference on Research and Development in Information Retrieval*, London, England, 1994.

[53] H. S. Seung, M. Opper and H. Sompolinsky, "Query by Committee," in *Proceedings of the 5th Annual ACM Conference on Learning Theory (COLT)*, Pittsburgh, PA, 1992.

[54] G. Cormack and T. R. Lynam, "Text Retrieval Conference (TREC) 2007 Public Spam Corpus," 9 November 2007. [Online]. Available: http://plg.uwaterloo.ca/~gvcormac/treccorpus07/. [Accessed 20 September 2013].

[55] L. C. Freeman, "Centrality in Social Networks: Conceptual Clarification," *Social Networks*, vol. 1, no. 3, pp. 215-239, 1979.

[56] S. Wasserman and K. Faust, "Graphs and Matrices," in *Social Network Analysis: Methods and Applications*, Cambridge University, United Kingdom, Cambridge University Press, 1994, pp. 92-166.

[57] P. Calais, D. Guedes, W. Meira Jr, C. Hoepers, M. Chaves and K. Steding-Jessen, "Spamming Chains: a New Way of Understanding Spammer Behavior," in *Proceedings of the 6th Conference on E-mail and Anti-Spam (CEAS)*, Mountain View, CA, 2009.

[58] M. Sahami, S. Dumais, D. Heckerman and E. Horvitz, "A Bayesian Approach to Filtering Junk E-mail," in *Proceedings of the American Association for Artificial Intelligence (AAAI) Workshop on Learning for Text Categorization*, Madison, WI, 1998.

[59] D. Chinvale, P. Kolari, T. Oates and T. Finin, "Ensembles in Adversarial Classification for Spam," in *Proceedings of the 18th Association for Computing Machinery (ACM) Conference on Information and Knowledge Management (CIKM)*, Hong Kong, China, 2009.

[60] E. Bingham and H. Mannila, "Random Projection in Dimensionality Reduction: Applications to Image and Text Data," in *Proceedings of the 7th International Conference on Knowledge Discovery and Data Mining (KDD)*, San Francisco, CA, 2001.

[61] A. Blum, "Random Projection, Margins, Kernels, and Feature Selection," in *Pattern Analysis, Statistical Modeling, and Computational Learning (PASCAL) Statistical and Optimization Perspectives Workshop on Subspace, Latent Structure, and Feature Selection (SLSFS) Techniques*, Bohinj, Slovenia, 2005.

[62] S. Dasgupta and A. Gupta, "An Elementary Proof of a Theorem of Johnson and Lindenstrauss," *Randomness Structures & Algorithms,* vol. 22, no. 1, pp. 60-65, 2003.

[63] C. Cortes and M. Mohri, "Confidence Intervals for the Area Under the Receiver Operating Characteristic Curve," in *Proceedings of the 18th Advances in Neural Information Processing Systems (NIPS) Conference*, Vancouver, British Columbia, Canada, 2005.

[64] M. Barreno, B. Nelson, A. D. Joseph and J. D. Tygar, "The Security of Machine Learning," *Machine Learning,* vol. 81, no. 2, pp. 121-148, 2010.

[65] N. Dalvi, P. Domingos, Mausam, S. Sanghai and D. Verma, "Adversarial Classification," in *Proceedings of the 10th International Conference on Knowledge Discovery and Data Mining (KDD)*, Seattle, WA, 2004.

[66] A. Globerson and S. Roweis, "Nightmare at Test Time: Robust Learning by Feature Deletion," in *Proceedings of the 23rd International Conference on Machine Learning (ICML)*, Pittsburgh, PA, 2006.

[67] C. Thiel, "Classification on Soft Labels is Robust Against Label Noise," in *Proceedings of the 12th International Conference on Knowledge-Based Intelligent Information and Engineering Systems (KES)*, Zagreb, Croatia, 2008.

[68] O. Dekel, O. Shamir and L. Xiao, "Learning to Classify with Missing and Corrupted Features," *Machine Learning,* vol. 81, no. 2, pp. 149-178, 2010.

[69] H. Xiao, H. Xiao and C. Eckert, "Adversarial Label Flips Attack on Support Vector Machines," in *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI)*, Montpellier, France, 2012.

[70] B. Nelson, M. Barreno, F. J. Chi, A. D. Joseph, B. I. P. Rubinstein, U. Saini, C. Sutton, J. D. Tygar and K. Xia, "Exploiting Machine Learning to Subvert Your Spam Filter," in *Proceedings of the 1st USENIX Workshop on Large Scale Exploits and Emergent Threats (LEET)*, San Francisco, CA, 2008.

[71] C. Cortes and V. Vapnik, "Support Vector Networks," *Machine Learning,* vol. 20, no. 3, pp. 273-297, 1995.

[72] B. Scholkopf, A. J. Smola, R. C. Williamson and P. L. Bartlett, "New Support Vector Algorithms," *Neural Computation,* vol. 12, no. 5, pp. 1207-1245, 2000.

[73] M. A. Fischler and R. C. Bolles, "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography," *Communications of the Association for Computing Machinery (CACM),* vol. 24, no. 6, pp. 381-395, 1981.

[74] L. Xu, E. Oja and P. Kultanen, "A New Curve Detection Method: Randomized Hough Transform (RHT)," *Pattern Recognition Letters,* vol. 11, no. 5, pp. 331-338, 1990.

[75] R. Segal, G. Cormack and A. Bratko, "Conference on E-mail and Anti-Spam (CEAS) 2008 Live Spam Challenge Corpus," 22 August 2008. [Online]. Available: http://plg.uwaterloo.ca/~gvcormac/ceascorpus/. [Accessed 20 September 2013].

[76] S. Abu-Nimeh, D. Nappa, X. Wang and S. Nair, "A Comparison of Machine Learning Techniques for Phishing Detection," in *Proceedings of the 2007 Anti-Phishing Working Group (APWG) eCrime Researchers Summit*, Pittsburgh, PA, 2007.

[77] J. Nazario, "Phishing Corpus Mailbox 2," 7 August 2006. [Online]. Available: http://monkey.org/~jose/phishing/phishing2.mbox. [Accessed 20 September 2013].

[78] D. M. Blei, A. Y. Ng and M. I. Jordan, "Latent Dirichlet Allocation (LDA)," *Journal of Machine Learning Research (JMLR),* vol. 3, no. 1, pp. 993-1022, 2003.

[79] Stanford Natural Language Processing Group, "Topic Modeling Toolbox," 4 December 2011. [Online]. Available: http://nlp.stanford.edu/software/tmt/. [Accessed 20 September 2013].

[80] A. Y. Ng, M. I. Jordan and Y. Weiss, "On Spectral Clustering: Analysis and an Algorithm," in *Proceedings of the 14th Advances in Neural Information Processing Systems (NIPS) Conference*, Vancouver, British Columbia, Canada, 2001.

[81] F. R. K. Chung, Spectral Graph Theory, Providence, RI: American Mathematical Society, 1996.

[82] K. S. Xu, M. Kliger and A. O. Hero III, "Identifying Spammers by Their Resource Usage Patterns," in *Proceedings of the 7th Annual Conference on E-mail and Anti-Spam (CEAS),* Redmond, WA, 2010.

[83] Unspam Technologies, "Project HoneyPot," 2010. [Online]. Available: http://www.projecthoneypot.org/. [Accessed 20 September 2013].

[84] J. Nazario, "Phishing Corpus," 7 August 2007. [Online]. Available: http://monkey.org/~jose/wiki/doku.php. [Accessed 20 September 2013].

[85] Apache SpamAssassin Project, "SpamAssassin Corpus," 31 January 2006. [Online]. Available: http://spamassassin.apache.org/publiccorpus/. [Accessed 20 September 2013].

[86] O. Chapelle, B. Scholkopf and A. Zien, "A Discussion of Semi-Supervised Learning and Transduction," in *Semi-Supervised Learning*, Cambridge, MA, Massachusetts Institute of Technology (MIT) Press, 2006, pp. 457-462.

[87] T. Joachims, "Transductive Learning via Spectral Graph Partitioning," in *Proceedings of the 20th International Conference on Machine Learning*, Washington, DC, 2003.

[88] B. T. Adler, L. de Alfaro, S. M. Mola-Velasco, P. Rosso and A. G. West, "Wikipedia Vandalism Detection: Combining Natural Language, Metadata, and Reputation Features," in *Proceedings of the 12th International Conference on Intelligent Text Processing and Computational Linguistics (CICLing)*, Tokyo, Japan, 2011.

[89] H. He and E. A. Garcia, "Learning from Imbalanced Data," *Institute for Electrical and Electronics Engineers (IEEE) Transactions on Knowledge and Data Engineering*, vol. 21, no. 9, pp. 1263-1284, 2009.

[90] R. Klinkenberg, "Learning Drifting Concepts: Example Selection versus Example Weighting," *Intelligent Data Analysis*, vol. 8, no. 3, pp. 281-300, 2004.

[91] R. O. Duda, P. E. Hart and D. G. Stork, "Bayesian Decision Theory," in *Pattern Classification, 2nd Edition*, New York, NY, John Wiley & Sons Inc, 2001, pp. 20-83.

[92] P. Domingos, "MetaCost: A General Method for Making Classifiers Cost Sensitive," in *Proceeding of the 5th International Conference on Knowledge Discovery and Data Mining (KDD)*, San Diego, CA, 1999.

[93] N. V. Chawla, K. W. Bowyer, L. O. Hall and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321-357, 2002.

[94] E. B. Wilson, "Probable Inference, the Law of Succession, and Statistical Inference," *Journal of the American Statistical Association (ASA),* vol. 22, no. 158, pp. 209-212, 1927.

[95] B. Caputo, K. L. Sim, F. Furesjo and A. J. Smola, "Appearance-Based Object Recognition Using SVMs: Which Kernel Should I Use?," in *Proceedings of the Neural Information Processing Systems (NIPS) Workshop on Statistical Methods for Computational Experiments in Visual Processing and Computer Vision*, Whistler, British Columbia, Canada, 2002.

[96] C. Chen, A. Liaw and L. Breiman, "Using Random Forests to Learn Imbalanced Data," University of California at Berkeley, Berkeley, CA, 2004.

[97] C. Phua, "Minority Report Data," 2007. [Online]. Available: https://sites.google.com/site/cliftonphua/minority-report-data.zip. [Accessed 20 September 2013].

[98] A. Gepp, J. H. Wilson, K. Kumar and S. Bhattacharya, "A Comparative Analysis of Decision Trees Vis-a-vis Other Computational Data Mining Techniques in Automotive Insurance Fraud Detection," *Journal of Data Science,* vol. 10, no. 3, pp. 537-561, 2012.

[99] K. Bache and M. Lichman, "Statlog German Credit Risk Data Set," 1994. [Online]. Available: http://archive.ics.uci.edu/ml/datasets/Statlog+%28German+Credit+Data%29. [Accessed 20 September 2013].

[100] C. Elkan, "KDD Cup 1999 Computer Network Intrusion Detection Data Set," 1999. [Online]. Available: http://www.sigkdd.org/kdd-cup-1999-computer-network-intrusion-detection. [Accessed 20 September 2013].

BIOGRAPHY

David DeBarr graduated from Cary Senior High School in Cary, North Carolina, in 1986. He received his Bachelor of Science degree (summa cum laude) from the University of Maryland in 1996. He received his Master of Science degree from George Mason University in 2006. Previously employed as an Applied Researcher at Microsoft, he currently works as a Computer Scientist at the MITRE Corporation in McLean, VA.