8-8-2009

Using random projections for dimensionality reduction in identifying rogue applications

Travis Levestis Atkison

USING RANDOM PROJECTIONS FOR DIMENSIONALITY REDUCTION IN IDENTIFYING ROGUE APPLICATIONS

By

Travis Levestis Atkison, Jr.

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

Mississippi State, Mississippi

August 2009

USING RANDOM PROJECTIONS FOR DIMENSIONALITY REDUCTION

IN IDENTIFYING ROGUE APPLICATIONS

By

Travis Levestis Atkison, Jr.

Approved:

Rayford B. Vaughn, Jr. Professor of Computer Science and Engineering (Major Professor)	David A. Dampier Associate Professor of Computer Science and Engineering (Committee Member)
Mahalingam Ramkumar Assistant Professor of Computer Science and Engineering (Committee Member)	Yoginder Dandass Assistant Professor of Computer Science and Engineering (Committee Member)
Kirk Arnett Professor of Management and Information Sciences (Committee Member)	Edward B. Allen Associate Professor and Graduate Coordinator of Computer Science and Engineering

Sarah A. Rajala
Dean of Bagley College of Engineering

Name: Travis Levestis Atkison, Jr.

Date of Degree: August 8, 2009

Institution: Mississippi State University

Major Field: Computer Science

Major Professor: Dr. Rayford B. Vaughn, Jr.

Title of Study: USING RANDOM PROJECTIONS FOR DIMENSIONALITY REDUCTION IN IDENTIFYING ROGUE APPLICATIONS

Pages in Study: 124

Candidate for Degree of Doctor of Philosophy

In general, the consumer must depend on others to provide their software solutions. However, this outsourcing of software development has caused it to become more and more abstract as to where the software is actually being developed and by whom, and it poses a potentially large security problem for the consumer as it opens up the possibility for rogue functionality to be injected into an application without the consumer's knowledge or consent. This begs the question of 'How do we know that the software we use can be trusted?' or 'How can we have assurance that the software we use is doing only the tasks that we ask it to do?' Traditional methods for thwarting such activities, such as virus detection engines, are far too antiquated for today's adversary. More sophisticated research needs to be conducted in this area to combat these more technically advanced enemies.

To combat the ever increasing problem of rogue applications, this dissertation has successfully applied and extended the information retrieval techniques of *n*-gram analysis

and document similarity and the data mining techniques of dimensionality reduction and attribute extraction. This combination of techniques has generated a more effective Trojan horse, rogue application detection capability tool suite that can detect not only standalone rogue applications but also those that are embedded within other applications. This research provides several major contributions to the field including a unique combination of techniques that have provided a new tool for the administrator's multi-pronged defense to combat the infestation of rogue applications. Another contribution involves a unique method of slicing the potential rogue applications that has proven to provide a more robust rogue application classifier. Through experimental research this effort has shown that a viable and worthy rogue application detection tool suite can be developed. Experimental results have shown that in some cases as much as a 28% increase in overall accuracy can be achieved when comparing the accepted feature selection practice of mutual information with the feature extraction method presented in this effort called randomized projection.

Key words: rogue application detection, information retrieval, data mining, *n*-gram analysis, clustering, dimensionality reduction, randomized projection

DEDICATION

I would like to dedicate this dissertation to my beautiful wife, Rebekah, and my wonderful children, Anthony and Zachary, whose unending love and support have helped make this a reality.

ACKNOWLEDGMENTS

My first acknowledgment, without question, must be to thank my Lord and Savior, Jesus Christ, for without Him we would not be here. Philippians 4:13 says that "I can do everything through Him who gives me strength." Without Him constantly and continually providing me my daily strength, this would not have been possible.

> The LORD is my strength and my song; He has become my salvation. He is my God, and I will praise him, my father's God, and I will exalt him.
>
> –Exodus 15:2

> He put a new song in my mouth, a hymn of praise to our God. Many will see and fear and put their trust in the LORD... But may all who seek you rejoice and be glad in you; may those who love your salvation always say, "The LORD be exalted!"
>
> –Psalms 40:3, 16

I would like to express my sincerest appreciation to my advisor Dr. Rayford Vaughn. The research opportunities and generous financial support he has provided have allowed me to pursue my and my family's dreams. The kindness and sincerity that he and the entire faculty have shown us could never be repaid. That sincerity, first shown when we met Dr. Vaughn and Dr. David Dampier during our campus visit in April 2005, was one of the main factors in our decision to choose Mississippi State University to pursue this degree. I would also like to thank Dr. Julia Hodges for the wonderful teaching opportunities that she has provided. It affirmed my chosen career path and gave me tremendous confidence

that I can continue to be a successful teacher. Furthermore, I would like to thank my entire committee for their time, patience, and thought provoking discussions throughout this process. I would also like to thank the National Science Foundation who partially supported this work under grant SCI0430354 04090852.

On a personal note, I must pay the utmost respect to my many friends and family members whose support throughout this endeavor has been limitless. They will never truly understand the gratitude and love that I have for them. In particular there are two dear friends and their families I want to give special thanks. The first is Dr. Randy Smith, his wife Jennifer and their two terrific boys Benjamin and Maxwell. For the last 15 plus years Randy and his family have been there for me and my family in the highest of highs and lowest of lows. Their willingness to be there for us over the years, whether in helping us move or in wise counsel, can never be thanked enough. Randy is like a brother to me, and his family is an integral part of our family. The second is Dr. Chad Steed, his wife Jessica and their two wonderful children, Julia and Blake. The last four years have been a true blessing having Chad and his family in our lives. The ability to be there and lean on each other while going through this Ph.D. process was tremendous and absolutely one of the main reasons that we have been successful in this pursuit. Their constant lovingness and faith has been an inspiration for me and my family. Friends like Randy and Chad are hard to come by, and I treasure and know that God has placed them in our lives. I am truly thankful for both of them.

To my parents, there are no words I can use to do justice in describing what I want to say. No matter what the circumstance, you both have been there for me to provide

endless and unconditional support, guidance and love. Your sacrifices have been immeasurable. You showed me, through your words and actions, that with hard work I can accomplish my goals. You laid the groundwork that has allowed me to reach this point. To say thank you is not nearly enough.

Finally, I must say a special thanks to my beautiful wife Rebekah, who truly is the light of my life. We began our journey twelve years ago, and from the beginning you have been my rock. Thank you for putting up with my orneriness over these past few years. Without your undying love and support I would not have made it through. To my two fantastic boys, Anthony and Zachary, you have been a true blessing in my life. Thank you for always being there and giving me loving hugs and kisses when I needed them the most. I praise the LORD everyday that he has placed us together and blessed us with His eternal love.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In general, a consumer, whether corporate or private, must depend on some other entity to deliver their software needs. This software can come from several different sources ranging from software development firms to downloading freeware from the Internet. These software needs include, but are not limited to, the operating system, virus detection engines, firewall systems, or the latest database software to hold consumer music files. Even specialized applications for corporate business needs are often purchased rather than developed "in-house." The outsourcing of application development coupled with the globalization of the software development market means that where software is being developed and by whom is becoming more and more abstract. This scenario presents a large security problem for the consumer in that the developers of these computing applications may not have the same philosophies or views as the users of these applications. An example scenario might be as follows: a virus detection developer provides a consumer with a detection engine that is programmed to ignore certain viruses and labels the machine as "clean" or free of viruses. Another example could involve a firewall software developer manipulating the consumer's firewall application to report to an outside entity information regarding network traffic data that passes through the firewall,

without the consumer's knowledge or consent. This information could be as benign as the website that the consumer is visiting all the way to actions such as what database queries the users are making and their results. The latter example is known as an "information leakage vulnerability attack."

Information leakage can be defined as when "non-public" information is released (or leaked) without the information owner's knowledge or consent. This type of vulnerability can be introduced within an application at design time through malicious intent or through poor programming practices. It can also be introduced by a rogue attacker after deployment by being bundled with, or concealed within, a non-threatening application. Symantec reported in their bi-annual threat report for the first half of 2005 that "six of the top ten spyware (information leakage) programs were delivered to their victim by being bundled with some other program." [65]

I label these vulnerabilities, whether with or without malicious intent, as rogue functionalities and the executables that contain rogue functionality as rogue applications. Whether the applications have malicious intent or not, rogue applications are "defined to be programs that perform a malicious function, such as compromising a system's security, damaging a system or obtaining sensitive information without the owner's informed consent." [2,59] The concept of rogue applications that have rogue functionalities is not a new idea. The first accounts of the existence of rogue applications can be dated back to 1949 when John von Neumann postulated that a computer program could replicate itself. [71] However, the problem of accurately and effectively detecting and eradicating these rogue applications is still a difficult research problem.

1.1 Motivation

Traditionally, rogue applications are categorized into the following three groups based primarily on their delivery mechanisms: viruses, worms and Trojan horses. A virus cannot propagate throughout the network to reach its victims on its own; some action is required by the user to activate the virus, for example execution of an application on the target system. On the other hand a worm is self-contained and can transport itself over the network to its victims usually through some known vulnerability in the victim's computer that has yet to be dealt with through updating or patching. Lastly, a Trojan horse takes on the appearance of a benign application but underneath actually performs some rogue functionality, for example information leakage. [44] Although they arrive at the final destination in different manners, all three variations exist to do harm to a user's computer or disrupt the user's activities.

To combat these attacks on a system, industry as well as the home consumer has turned to anti-virus software which contain virus detection engines. "A large percentage of the security software industry is built on the practice of looking for the digital patterns (signatures) that identify known threats." [51] According to the 2007 Computer Security Institute (CSI) Computer Crime and Security Survey, anti-virus software, which is a signature based solution, accounted for 54.3 percent of the total budget for industry software security in 2005. [51] Though very good at what they do, virus detection engines rely on a database of signatures to detect known rogue applications. These signatures are "case-specific features extracted from viruses in order to detect those same instances in the future" [26] and must be updated each time a new and previously unknown rogue

application surfaces. The signatures attempt to capture the syntactic characteristics and therefore are vulnerable to obfuscation techniques. [48] The creation of these signatures is a manual and time consuming activity. Each potential rogue application must be first gathered, which is difficult in and of itself, then analyzed and dissected to determine if it in fact has a malicious intent. If a malicious intent is found, the analyst must find an identifying signature that can be used by the detection engine for detecting it in the wild. These solutions are far too antiquated for today's adversary. According to CSI "criminals have pushed the state of [rogue software] to a point where signature detection is less and less effective." [51] It also has to be noted that adversaries have access to all of the modern anti-virus tools and can easily modify their rogue application to overt detection.

Signature based systems inherently limit the detection of new and previously unknown types of rogue attacks. To make this method more difficult, rogue application writers are getting much better at disguising or obfuscating their applications. Obfuscation is the act of hiding the meaning of something from view. In the case of application code, obfuscation is done to prevent reverse engineering of the application. Obfuscation is such a powerful technique that Christodorescu noted in his comparison of three popular commercial virus detection engines that they "could be subverted by very simple obfuscation transformations." [15] With the introduction of rogue application writing kits freely available on the Internet anyone can download these and become a rogue application writer instantly. This easy access exponentially increases the amount of rogue applications that are in the wild for signature writers to analyze. To try to combat this limitation, some of the more modern anti-virus software systems employ heuristics based detection. Heuris-

tic detection tries to identify a rogue application by partially running the beginning of the application. If no rogue functionality has been detected during the partial run, control of the application is turned over to the system. Even though these systems claim they can detect new and previously unknown attacks, these methods still have their limitations. For example, "both Network Associates' McAfee VirusScan and Symantec's Norton Antivirus *missed* the Melissa virus completely with their heuristics scanners." [62]

The purpose of this research was to help answer the important software assurance question of 'How do we know that the software we are using is doing what we asked it to do and more importantly nothing more?' The main focus of this research effort has been to design and develop a process that aids in the detection of rogue applications by combining techniques from the information retrieval and data mining fields. Particular concentration has been given to Trojan horses as described above.

Previous attempts [26, 36, 37, 50] to use methodologies and techniques from information retrieval and data mining have had some success but all have come against the 'curse of dimensionality.' The 'curse of dimensionality,' first referred to by Bellman [10], generally describes the computational issues related to performing mathematical operations within an extremely high dimensional space. These dimensions can number in the 10^8 and higher range. Therefore, a capability to reduce the number of features to a more manageable number is very useful in countering this problem. Making any decisions based on this high-dimensional data will require the construction of a low-dimensional embedding that preserves the underlying "structure" hidden in the data. For this research I used a technique called randomized projection [40, 42] to create the low-dimensional

embeddings. This work has shown that these randomized projection algorithms can guarantee that the distances between points in the original data set remain almost invariant in the projected data set, which usually has a dramatic reduction in the number of dimensions.

It is my belief that a multi-pronged defense approach where there are several weapons in the administrator's toolbox is needed to attack this problem. Developing a more effective rogue application detection tool through the use of information retrieval and data mining methodologies, as well as dimensionality reduction through randomized projections will provide one more weapon in the consumer's toolbox to attack this problem. No one is immune from these malicious attacks; from the corporation to the unsuspecting home user, everyone is at risk. Therefore, an entire range of consumers can benefit from this research effort.

1.2 Statement of Hypothesis

In this dissertation research I have applied and extended the information retrieval techniques of *n*-gram analysis and document similarity and the data mining techniques of dimensionality reduction and attribute selection. The overall hypothesis of this research was that rogue application detection could be accomplished by cleverly combining information retrieval and data mining techniques. The specific components of this hypothesis that I have tested are:

1. *By applying and extending the information retrieval technique of n-gram analysis and the data mining technique of attribute extraction, a Trojan horse, rogue application detection capability tool can be developed that will provide a true positive rate above 98% and false positive rate below 3%.*
2. *By combining information retrieval and data mining techniques in concert with the dimensionality reduction technique of randomized projection, a more accurate*

solution in detecting rogue attacks can be developed when compared to seminal efforts currently used in research today. A more accurate solution is defined in terms of a 3% increase in overall accuracy when comparing this method to that of the mutual information dimensionality reduction method.

In summary, this research has focused on obtaining predictive features from applications using *n*-grams, performing feature extraction using the technique of randomized projection and creating a prediction capability using cosine similarity techniques and methodologies. This research has shown that by using the power of randomized projections to create an accurate low-dimensional embedding of the original data, an ability to better use the power of the prediction algorithms will be gained. With the results of this research effort, which I will present below, I claim that this technique provides a rogue application detection capability that is "better" than the current popular mutual information technique by comparing my results to those of Kolter's. For this research "better" has been defined in terms of performance, measured using standard validation performance factors.

1.3 Contributions

With the successful completion of this research I have produced a unique detection capability that has positively augmented the current capabilities to provide for greater coverage in the fight against rogue applications. Specifically the main practical contributions of this research are:

- Extension of rogue application detection capabilities that enable greater protection of consumer systems
- Development of a rogue detection capability that successfully uses aspects of information retrieval and data mining

- Successful validation of the usefulness of the combination of data mining and information retrieval techniques as applied to the rogue application detection problem
- New to rogue application detection literature, an approach that extracts the code and data sections from an application and thereby produces a higher accuracy detection tool
- Most importantly, the development of a unique method for selecting attribute sets using the dimensionality reduction technique of randomized projection from target data for greater success rate in detecting rogue applications

1.4 Organization

The remainder of this proposal is organized as follows. Chapter 2 provides an overview of the current literature on this research topic. Chapter 3 provides the experimental plan that was conducted for this research, including short descriptions of the research components for the rogue application detection capability. Chapter 4 provides the experimental results and analysis of both the preliminary proof of concept and the full experiment using a large data set. Chapter 5 concludes this dissertation with future work and both completed and expected publications.

CHAPTER 2

LITERATURE REVIEW

It has been proven that distinguishing between an application that has rogue code embedded in it and an application that does not is intractable in the general case. [5] See Appendix A for the formal proof provided by Adleman. Adleman's proof shows that to detect all possible viruses in a given application reduces to the halting problem. However, this does not discount the idea that generic decisions for detecting broad classes of rogue functionality can be made.

This research used the generic decision basis to explore and evaluate the effectiveness of using a combination of information retrieval and data mining techniques, with the concentration on dimensionality reduction, as a solution to part of the rogue application detection problem. As described in the motivation section of the introduction, the identification and detection of rogue applications was the tract of this research effort. There are two high-level major pathways of exploratory research into solving the rogue application detection problem. These are static analysis and dynamic analysis. I delineate these two methods in the following manner. A dynamic analysis technique involves the actual execution of the potential rogue application to detect if in fact rogue functionality is taking place within the executable. On the other hand I define static analysis techniques

as making the rogue functionality determination without having to execute the application. These two camps can not solve this problem alone but only in concert can the total solution be derived.

This research effort was not the first to try to solve the important topic of detecting and classifying new or previously unknown rogue applications. Presented in the remainder of this chapter is a survey of the most significant research that is related to the efforts discussed in this paper.

2.1 Static Analysis

The advantage of static analysis in the detection of rogue software applications is that detection is made without having to execute the application and risk harm or contamination to the network from any potential rogue functionality embedded within it. Below are descriptions of the major research efforts that have used various methods of static analysis to solve this detection problem.

2.1.1 General

In an early work Lo et al. [41] described their tool known as the Malicious Code Filter (MCF). They developed a "novel approach to distinguish malicious code from benign programs" using what they identify as "tell-tale signs." According to Lo, "a tell-tale sign is a program property that allows us to determine whether or not a program is malicious without requiring a programmer to provide a formal specification." [41] Some of these "tell-tale" signs that their filter looks for include: file reading and writing, program

execution, anomalous data flow and authentication. [41] In identifying a Trojan login application they stated that it was "detectable with the 'Authentication' tell-tale sign." [41] It has to be noted that Lo et al. does not identify which Trojan horse application their tool was able to detect or provide any experimental results to validate their claims. They simply state that their tool is successful on the small data set that they were using.

Though not claiming to use information retrieval methodologies, Christodorescu et al. [16] do similar work using 'templates' which contain 'instruction sequences' of rogue functionalities. Their work, then, is to analyze a target application and determine the similarity of segments of the target application to their templates. The similarity is determined "if there exists an assignment to variables from the template node expression that unifies it with the program node expression." [16]

2.1.2 Information Retrieval Related

Information retrieval, traditionally, is the "part of computer science which studies the retrieval of information (not data) from a collection of written documents." [8] These retrieved documents' aim is to "satisfy a user's information need." [8] One of the more notable uses of information retrieval has been in the field of document authorship. Here the "information need" is to determine whether or not a particular person(s) authored a particular document or collection of documents. There are several popular examples in this area including determining if William Shakespeare actually authored the works attributed to him [27], did Hamilton or Madison write the 12 disputed Federalist papers [13, 24] or did Paul write all the books of the Christian New Testament Bible attributed to him [53].

These techniques have been used more recently in the area of software forensics to determine application authorship [21, 25, 54, 64] as well as determining programming plagiarism [14, 70]. The idea of using clues from one item to make a determination on another or fulfill an "information need" is not a foreign concept. Therefore making the leap of using this methodology to determine whether or not a particular application has rogue functionality is not foreign.

As with any of the techniques in either information retrieval or data mining below, feature selection is pivotal to the detection/prediction success of the method. There have been several efforts [3, 4, 32, 36, 37, 43, 50, 74] that look at using the information retrieval concept of n-grams as a potential technique for feature selection. An n-gram is "any substring of length n." [8] In most of these efforts the gram (which will be the composite of the substring) is a byte in hexadecimal form.

Abou-Assaleh et al. used the Common N-Gram (CNG) analysis method [33] which has been used in authorship attribution and text clustering to detect rogue applications. [3, 4] "The CNG classification method relies on profiles for class representation." [3, 4] The most frequent n-grams represent a class profile with a parameterized length. [3, 4] To overcome the computational complexity that goes with the large number of n-grams produced from a data set, Abou-Assaleh et al. "limited" the number of n-grams to those that had the highest frequencies. [3] Once they had their feature vectors, they used the traditional information retrieval method k-nearest neighbors, with $k = 1$, for their classification. Although the authors never mention the size of their data set in terms of number of executables, they do note that it consists of Windows binary executables

that contain worms and viruses. They experimented with several variables including the length of the *n*-grams, the maximum number of *n*-grams to consider as well as the size of the CNG profiles with a reported highest detection rate of 94%. [3] Although this sounds like a good number, there are other factors that need to be considered. One of these factors that the authors do not mention is the false positive rates for the experiments. This is very important because if their false positive rate was 50%, meaning that their solution identified 1 out of 2 applications as rogue, then a 94% detection rate is meaningless. Another factor to consider is that the authors state that the 94% detection rate experiment had some of the training data mixed in with the testing data. [3] Unfortunately, this biasing of the testing data cast some doubt on these results. Though issues exist with their experimental results the methodology is sound and can be built upon.

Marceau [43] puts an interesting twist on the problem of using *n*-grams as features by having "multiple-length" grams instead of the traditional single *n*-length gram. To accomplish this he first creates a suffix tree of the *n*-grams where *n* is 'large enough' and then "compacts the suffix tree to a directed acyclic graph (DAG) by merging 'equivalent' subtrees." [43] By doing this Marceau has a set of grams with varying lengths that is "equivalent to the original set of *n*-grams." [43] From this suffix tree the authors create a Finite State Machine (FSM) for their "two-finger algorithm" which is their classifier. This algorithm is very similar to a sliding window algorithm. Here each "move of a finger" will correspond to a branch in the suffix tree or a state on the FSM. The authors presented results of their algorithm on three different data sets (the *lpr* and *inetd* data from the University of New Mexico and data from the CORBA Immune System) as compared

to the sliding window work of Forrest. [23] The results of the comparison were very similar but the advantage went to the "two-finger algorithm" because of the far less space needed to store the feature sets. Although these results are promising, the authors note that they are preliminary. [43] Additional work needs to be done to see if the additional computational complexity trade-off for computing the suffix tree and corresponding FSM is worth the gains in space reduction.

Zhang et al. [74] propose a new method of reducing the feature set dimension by using rough set theory. They narrow their work by focusing exclusively on *n*-grams of size 2 bytes. Once the 2-byte grams have been extracted the authors select their features based on mutual information. This is the same method that Kolter [36, 37] (see Section 2.1.4) uses. However, Zhang et al. further reduce the feature set dimension by using Rough Set Theory. [74] They look at the fundamental concepts of "reducts and cores." Because of the computation overhead associated with calculating the relative core, the authors developed "an efficient implementation based on positive region definition." [74] This allows them to quickly reduce redundant attributes. Though the authors present some preliminary results on a small Windows PE data set using a clustering algorithm, the main focus of their research was to present a novel feature reduction methodology using Rough Set Theory.

2.1.3 Data Mining Related

Along the same timeframe as Lo's et al. work described above there was an effort ongoing at IBM's T. J. Watson Research Center by Kephart et al. [32] to use "biologically

inspired anti-virus algorithm" methods to detect and defend against rogue applications. This work concentrated on boot sector viruses using both single-layer and "hand-crafted" multi-layer neural networks. The key element in this work, as with any of these approaches, is the feature selection. Kephart et al. "extracted a set of 3-byte strings or 'trigrams' that appeared frequently in boot sectors of viruses but infrequently in legitimate ones." [32] These 'trigrams' were all hand created after human examination of boot sectors in their data set, which was about 150 boot sectors. Their work resulted in a neural network that had a detection rate of approximately 85% on their test data set. [32] The result is good and not surprising considering the amount of pairing of features to fit within the parameters of their neural network. This pairing was done purely by population density of the particular 'trigram' with respect to the infected and benign test sets. An improvement on this technique was made by Tesauro et al. and incorporated into IBM's Anti-Virus software. [66] This work was able to improve the previous effort [32] and gain 100% accuracy rate of the benign boot sectors but still in the neighborhood of 80% to 85% on the target rogue boot sectors. [66] One of the possible shortcomings with this approach is the resource requirements, which the authors state are "considerable obstacles" in deployment in large commercial efforts. [66]

In one of the first 'major' efforts to use data mining techniques to detect and identify rogue applications, Schultz et al. moved away from boot sector viruses and used Naïve Bayes learners for detection. [59] For this effort Schultz and his group gathered approximately 4500 applications from various FTP sites and used McAfee's virus scanner to label them as benign or rogue. [59] This produced a data set of approximately 3400 rogue ap-

plications and 1100 benign applications. Around 5% of their data were Trojan horses and the remaining 95% were viruses. [59] A subset of these applications was in the Windows Portable Executable (PE) format.

Schultz used a combination of three different feature extraction techniques: binary behavior profiling, string sequences and a binary transformation program *hexdump*, and three different learning algorithms: an inductive rule-based learner, a probabilistic method and a multiple classifier combiner system. [59]

For the first feature extraction technique they only used the subset of data that contained the Windows PE formatted binary files. They extracted three types of features using GNU's Bin-Utils tool suite. These were a list of Dynamic Linked Libraries (DLL's) used by the binary, a list of DLL function calls made by the binary and the number of different function calls made within each DLL. [59] Each one of these methods is composed of features that are combined into feature vectors of various lengths. The first method contained about 30 binary features, where the value of the feature was a 1 if present and a 0 if not present, and the second method contained approximately 2200 binary features, defined the same way as the first. The third method produced about 30 integer values that corresponded to the count of the number of times a particular function call was made within the application. [59]

The second and third feature extraction techniques looked at the entire data set. In the second feature extraction technique Schultz used the GNU Strings program to extract all sequences that contain printable characters from any location in the target applications. [59] Though these strings can be changed easily, they created binary feature vectors

that identified whether a particular string was present in the application or not. In the third and probably most robust feature extraction method the authors use, they create features of two-byte byte-sequences that were extracted from the applications using the *hexdump* tool. The thought process of using these techniques is that there will be similar sequences in rogue applications that are different than the similar sequences in the benign applications.

For their first feature set, Schultz et al. used an inductive rule learner called RIPPER [17, 18]. RIPPER "builds a set of rules that identify the classes while minimizing the amount of error." [59] Their rule set consisted of four rules for defining a rogue binary and "if it is inconsistent with all of the rogue binary hypotheses" [59] it is labeled benign. For the remaining two feature sets they used Naïve Bayes (used with the string features) and Multi-Naïve Bayes (used with the *hexdump* features) algorithms. They state that the Naïve Bayes rules were more complicated than the RIPPER rules which were just a "collection of the rules generated by each of the component Naïve Bayes classifiers." [59] Therefore the "prediction of the Multi-Naïve Bayes algorithm was the product of the predictions of the underlying Naïve Bayes classifiers." [59] Through their experiments Schultz concluded that the Multi-Naïve Bayes had the best detection rate but that for overall accuracy (factoring in false positive rate) that Naïve Bayes was best. Table 2.1 presents the cumulative results of their experiments.

Although these experiments show positive results, even the authors admit that they are far from perfect. One of the deficiencies is that the RIPPER algorithm was trained on a far smaller data set that the other algorithms. Therefore a true comparison between

the algorithms can not be made. Also, Schultz did not present an experimental result that combined the features from the three feature vector sets together into another feature set. Another potential shortcoming with these feature sets is their dependence on the strings, function names and DLL names extracted staying consistent over time. [59]

Table 2.1

Cumulative Results from Schultz [59]

Profile Type	**Detection Rate (%)**	**False Positive Rate (%)**	**Overall Accuracy (%)**
Signature Method - Bytes (*hexdump*)	33.75	0	49.28
RIPPER - DLLs used	57.89	9.22	83.62
RIPPER - DLL function calls	71.05	7.77	89.36
RIPPER - DLL counted function calls	52.63	5.34	89.07
Naïve Bayes - *strings*	97.43	3.80	97.11
Naïve Bayes - Bytes (*hexdump*)	97.76	6.01	96.88

2.1.4 Combination of Information Retrieval and Data Mining

One of the first and few works to combine the methodologies and techniques of information retrieval and data mining was by Kolter et al. [36, 37]. They extracted *n*-grams from their target dataset to create feature vectors for various data mining algorithms. Their data set had approximately 3800 applications gathered from various locations including applications from folders on Windows XP and 2000 operating systems, as well as downloaded from various websites such as download.com (www.download.com) and SourceForge (sourceforge.net). [36,37] About 2000 of these applications were benign and the remaining applications, gathered from VX Heavens (vx.netlux.org) contained various

viruses, worms and Trojan horses. [36,37] Unlike Schultz's work, all of these applications were in the Window's PE format.

Although the authors mention several methods for feature types, they concentrate on only *n*-grams, and more specifically *n*-grams that are four bytes in length. These four-byte sequences were extracted from their applications after the *hexdump* conversion tool had been applied converting each application into hexadecimal format. [36, 37] For each *n*-gram extracted from all applications Kolter created a long feature vector containing binary values of *true* if the particular *n*-gram was present in the application or *false* is it was not. [36,37] To pair down the overwhelming number of features they had, the authors "selected the most relevant attributes (i.e., *n*-grams) by computing the *mutual information* (*MI*)." [36,37] They used the following modified formula from Yang et al. [73] for mutual information:

$$MI(t) = \sum_{v_j \in \{0,1\}} \sum_{C_i} P(v_j, C_i) log \frac{P(v_j, C_i)}{P(v_j)P(C_i)} \tag{2.1}$$

Kolter defines each of the variables in their mutual information function as follows:

> C_i is the i^{th} class, v_j is the value of the j^{th} attribute, $P(v_j, C_i)$ is the proportion that the j^{th} attribute has the value v_j in the class C_i, $P(v_j)$ is the proportion that the j^{th} *n*-gram takes the value v_j in the training data, and $P(C_i)$ is the proportion of the training data belonging to the class C_i. [36, 37]

Kolter choose the top 500 *n*-grams produced by their mutual information algorithm as the optimum size of the feature vector. Then they applied the following data mining/machine learning algorithms: Instance-based learner, TFIDF (Term Frequency -

Inverse Document Frequency), Naïve Bayes, Support Vector Machines, Decision Trees and a Classifier Combining Algorithm. [36, 37]

Instance-based learners predict by finding an "example in the collection most similar to the unknown and return the example's class label as its prediction for that unknown." [36, 37] Kolter used a variant of this method that returned the k most similar making this algorithm just like k-nearest neighbor. [36, 37] The TFIDF is a traditional information retrieval vector space weighting model. To classify an unknown example, this weighting scheme, in conjunction with an information retrieval algorithm, computes a similarity of the unknown feature vector with each feature vector in the data set. "The method takes a weighted majority vote of the executable labels, and returns the class with the least weight as the prediction." [36] Naïve Bayes is a probabilistic method and Support Vector Machines is a linear classifier. [36, 37] Decision trees are a classical data mining technique where a tree is built by selecting attributes that produce the best splits in the training data set with exterior nodes defined as a class. [36, 37] Classifier combining is a majority vote of multiple decision trees that have been built with slightly different parameters. [36, 37]

Table 2.2 presents their predicted (P) versus actual (A) detection rate results for all the algorithms. The predicted rates are from tests performed on their entire data set, meaning that they used the entire data set to create the predictors. The actual detection rates are the results of applying these predictors to a smaller data set that has not been touched throughout the process. This was called their "real-world" data set. Kolter stated that even though the "Naïve Bayes performed much better" the "boosted decision tree

achieved the best overall performance in terms of best actual performance and matching the predicted performance." [37]

Table 2.2

Cumulative Results from Kolter [37]

Method	**Desired False Positive Rate (%)**					
	1		5		10	
	P (%)	**A (%)**	**P (%)**	**A (%)**	**P (%)**	**A (%)**
Boosted Decision Tree	94	86	99	98	100	100
Support Vector Machine	82	41	98	90	99	93
Boosted SVM	86	56	98	89	99	92
Instance-based, $k = 5$	90	67	99	81	100	99
Boosted Naïve Bayes	79	55	94	93	98	98
Decision Tree	20	34	97	94	98	95
Naïve Bayes	48	28	57	72	81	83

Though some of the results are interesting, there is room for improvement. The authors only look at *n*-grams for their feature set. They could have improved this work by combining *n*-grams and other features which should produce more robust predictions. Also, even the authors admit that their methods have "high computational overhead." [37] This is a detriment to any deployable solution but can be overcome with addition research and experimentation.

A typical use of *n*-grams as features can be seen in [26] where Henchiri et al. scan through every file for every *n*-gram, record their frequency and create a feature set. This obviously would create a list of features that is far too long and would make the computational complexity too high for any viable solution. To overcome this Henchiri imposed a "hierarchical feature selection process." [26] In addition to feature selection, they have

a feature elimination step in the hierarchy. In feature selection they specify an "intra-family support threshold of features within each virus family." [26] This "intra-family support threshold" means that if a particular *n*-gram is above this pre-defined threshold, the sequence is "retained as a candidate feature." A second threshold is placed in the feature elimination step. This threshold is for "inter-family support." The authors state that "this ensures that only those features that appear with a high enough inter-family support are retained." [26] This step might pose a problem in that it would get rid of certain features that are unique to a particular virus family. Results from using their feature extraction method with several machine learning algorithms were positive as compared to other more traditional methods. The authors admit the false positive rates are a little high but still manageable.

Reddy et al. [50] develop their own unique *n*-gram feature selection measure called 'class-wise document frequency.' This 'class-wise document frequency' is a variant on the traditional information retrieval concept of document frequency. Here the authors are looking at the number of applications in a particular pre-defined class that contain a certain *n*-gram. [50] This provides for a further refinement as well as lowering the number of features that exist for a particular class. Reddy also defines another concept called 'relevant *n*-grams.' The 'relevant *n*-grams' for a particular class of applications is the top *k* number of *n*-grams sorted on their frequency in the class. [50] Though they present a new method of feature selection the novel idea here is the further dissection of document frequency into a particular class frequency. To show that their feature selection method has merit Reddy used multiple machine learning/data mining classifiers (SVM, decision tree

and IBK) and combined them using the Dempster-Shafer Theory of Evidence. Again the novelty here is their method of combining the classifiers. The Dempster-Shafer Theory of Evidence "is an alternative to traditional probabilistic theory for the mathematical representation of uncertainty." [50, 60] It is a method for combining information from multiple sources and dealing with discrepancies. [50] They compare their work with that of Kolter and show slightly better results. The authors could improve this work by determining if their methods can be extracted to other rogue application classes.

2.2 Dynamic Analysis

Traditionally dynamic analysis of rogue software applications is done by running the potentially rogue application in a controlled/restricted environment, most times a virtual environment. Willems et al. define two different approaches to dynamic analysis of rogue applications:

- taking an image of the complete system state before rogue application execution and comparing it to the complete system state after execution
- monitoring the rogue application's actions during execution with the help of a specialized tool, such as a debugger [72]

Here an analyst would manually step through the application using a debugger or actively observe the results of the execution. As with static analysis above, this can be an extremely manual task and as noted there are far more rogue applications being created than can be examined.

Most tools that are currently in use only look at one execution path [9, 68]. The single execution path is the one that is executed in their virtual environment. Salois et

al. [55] compared several single path Commercial Off The Shelf (COTS) software tools, including PC-cillin and eSafe Protect Desktop, for dynamically detecting rogue applications, in particular a time bomb. A time bomb is a rogue application "that is triggered in a program when a specific logical condition relating to time is met." [55]

Bayer et al. developed a tool called TTAnalyze that "dynamically analyzes the behavior of Windows executables." [9] Though the authors developed TTAnalyze on top of a software emulator to reduce the footprint detectable by rogue applications it does not completely remove it; therefore, rogue code can detect that they are not being executed on the actual system. One way that Bayer admits this can happen is through "speed of execution" differences between the real system and their emulated one. [9] Once a rogue application can detect it is not being executed on a real system then potentially many of the analysis results are void. The analysis process of TTAnalyze is similar to other dynamic analysis techniques in that it executes the target application and collects all of the operating system and function calls made by the target application. The interesting research result is the authors accomplish this without 'hooking' the system or function calls. Instead they use 'page-directory base register' which "contains the physical address of the base of the page directory for the current process." [9] Though there were potentially positive results presented from TTAnalyze, it still is a single execution path tool and will potentially be susceptible to detection by the target rogue application.

Another single execution path method is presented by Vasudevan et al. [68] named Cobra. Cobra, which is actually a framework, is "facilitated by a technique that we call stealth localized-executions." [68] Basically, the authors state that the idea is to break

down the target application into 'blocks,' insert some 'invisible' Cobra specific code constructs and then execute the blocks. [68] Although the authors claim these inserted instructions go un-noticed by the rogue application, they did not publish any experiments that proved this is the case. The point of this effort was to show that the framework is efficient at the task of dynamically analyzing rogue applications and not its comparative performance to other potential solutions.

By only looking at a single execution path, a system does not take into account the existence of rogue functionality that has ties to a particular event, for example a date or a file condition. Moser et al. [46] proposes an effort to look at multiple execution paths at the same time. This effort will have a higher expectancy of finding the rogue functionality because the authors try to dissect all of the potential execution paths. To do this their analysis technique is "driven by the input that the program processes." [46] More precisely they check for places in the target application where a condition, for example branch, is determined based on some input and then take snapshots of the current state. Moser lists the following as input prospects: "reading the current time from the operating system, reading the content of a file or the result of a check for Internet connectivity." [46] If the target application performs one of these actions and then suddenly exits, the authors can back up to the previous snapshot, provide the action, and allow the target application to continue executing. [46] This seems to be a better method than just pursuing a single path but there are still issues that must be solved, namely the overhead associated with state saving and, as the author states, the problems created by 'external effects' such as "sending data over the network." [46] This work is still in its early stages and further

investigation is needed, noted by the fact that in their test data set their analysis did not follow multiple paths.

Moffie et al. [45] presented an effort that was geared directly toward detecting Trojan Horses called the Harrier Application Security Monitor. Moffie's approach attempts to stop the rogue application after it is on the victim's computer and started but before it is executed. This is a different pathway than above because instead of a dedicated analysis system the Harrier Application Security Monitor works in concert with the existing anti-virus software on the user's computer.

Others have suggested a solution that involves having a formal program specification for an application that spells out every intended behavior. [35] This in theory is a good idea but in practice is probably unrealizable on a large scale. By requiring written documentation that describes the behavior of an application the authors are introducing a complex and difficult to get accurate process. For the most part these techniques are looking at using system calls to detect the presence of rogue functionality when comparing the system audit logs to the formal specifications.

Yet another method of dynamic analysis involves running the target application and then analyzing the resulting audit logs for detection of rogue functionality. Rawat et al. [49] and Florez-Larrahondo et al. [22] provide examples of this type of detection method in their respective research efforts. Rawat applies a modified information retrieval method called a 'binary weighted cosine metric' as the similarity measurement for a standard k-nearest neighbor classifier to make a determination of rogue functionality. This 'binary weighted cosine metric' is an amalgam of the traditional cosine similarity

measure and a binary similarity measure that produces a similarity measure between two applications "using a metric that considers two factors; occurrence of system calls and the frequency of all system calls in the processes." [49] Though the results show promise, improvement can be obtained by further investigation using a data set that more closely matches current attack methodologies.

Florez-Larrahondo et al. looked at the problem of rogue application detection on distributed high-performance systems. Their solution called for a system of 'intelligent anomaly detection agents' to monitor both function and system call interfaces and feed this information to a neural network and a discrete Hidden Markov Model. [22] What is more interesting with this example versus Rawat is that the authors are using current data over a real system instead of simulated data. The authors note "the system may suffer from a large number of false positives because of difficulty collecting an adequate set of samples to train the models." [22] This is an issue with any potential solution in this realm not just this solution. Those not withstanding, the results presented are positive and show value in the research methodology.

2.3 Comparison of Static and Dynamic Methodologies

As can be seen, there have been several efforts using both static and dynamic approaches to solve this important research problem. There are also advantages and disadvantages to each method. When using a dynamic analysis methodology, whether analyzing audit logs or system calls, you are actually executing the potentially rogue application to make a prediction on intent. Because of the potential dangers involved, most

researchers will execute these potentially rogue applications in virtual environments, such as VMWare. However, many rogue application writers have designed in checks that can detect whether or not the rogue application is being executed inside of a virtual environment. If one is detected, the rogue code will skip over the harmful parts of the application, therefore, going unnoticed to the detection system. Also, recently Tom Liston and Ed Skoudis of Intelguardians, at the request of the Department of Homeland Security, have shown that a virtual environment can easily be circumvented and access to the host machine gained. This negates any 'security' a researcher had by using the virtual environment for rogue detection. Even with the potential pitfalls, dynamic analysis does provide real-time run-time analysis capabilities that static analysis can not provide.

Static analysis provides many advantages when attacking a problem as diverse and difficult as rogue application detection. Since the potential rogue applications are not executed there is no chance for accidental infestation as is the case with dynamic analysis. Though this is a positive, due to undecidability it is "impossible to certify statically that certain properties hold." [11] That aside, static analysis of potential rogue applications can be accomplished without the run-time overhead associated with dynamic analysis techniques. Using static analysis an analyst can discover all possible execution paths. Because of the ability to analyze applications without the need to execute them, this research has attacked the rogue application detection problem using static analysis techniques. This was done with the understanding that the solutions created here were only one piece of the larger solution that would include dynamic analysis.

2.4 Dimensionality Reduction using Randomized Projection

As can be seen from several of the potential solutions presented above, rogue application detection suffers from the problem that the data, once processed, is encoded in extremely high dimensions. This high-dimensional data, on the order of 10^8 or higher, limits the kind and amount of analysis that can be performed. Traditionally, there have been two categories of methods for dealing with the reduction of this type of high-dimensional data. These are feature selection and feature extraction. Though both have the net effect of reducing the dimensions of a given data set, I delineate these two methods in the following manner. Feature selection is merely a selection of a subset of the original feature set to produce the reduced dimension. Several examples of this type of dimensionality reduction can be seen in [36, 37, 59, 74]. The most notable example is Kolter's [37] mutual information. On the other hand feature extraction transforms, either linearly or non-linearly, the original feature set into a reduced set that retains the most important predictive information. Examples of this type include principle component analysis, latent semantic analysis and randomized projection.

There have been some efforts [12, 42, 47] that look at using randomized projection techniques for dimensionality reduction. "Randomized projection refers to the technique of projecting a set of points from a high-dimensional space to a randomly chosen low-dimensional subspace or embedding." [69] Minnila et al. [42] are using random projection techniques to map sequences of events and find similarities between them. Their specific application is in the telecommunication field looking at how to better handle network alarms. Their goal is to "show the human analyst previous situations that resemble

the current one" [42] so that a more informed decision about the current situation can be made. Though their proposed solution is not perfect, it does show the promise of using randomized projections in a similarity based application.

Arriaga et al. [6], in the field of concept learning, are using randomized projections to learn concept classes while maintaining a desired level of robustness in their thresholds. Their implementation is based on a neural network, which they call a neuronal, that allows for the robustness parameter not to be known in advance. Hristescu et al. [28] use the Smith-Waterman distance function in their randomized projection approach, called SparseMap, to perform efficient similarity searches of protein databases. Cowen et al. [19] are applying randomized projections in a pattern recognition problem where they are, among other methods, clustering Positron Emission Tomography scan brain volumes. Indyk et al. [29] as well as Kleinberg [34] were both able to show positive results using randomized projection in nearest neighbor searches.

Research that is more applicable to this dissertation work can be found in [12]. Bingham et al. apply randomized projections to an image and text retrieval problem. In comparison to this research problem their dimensions are not as large, 2500 for images and 5000 for text, but the results are still significant. The purpose of their work was to show that, compared to other more traditional dimensionality reduction techniques, such as principle component analysis or singular value decomposition, randomized projections offered a greater detail of accuracy. The authors were also able to show that there was significant computation saving by using randomized projections over other feature extraction techniques, such as principle component analysis.

In another text retrieval application, Kaski [31] successfully applied randomized projections in his text retrieval application that used WEBSOM, a graphical self-organizing map. Again Kaski turned to randomized projection as a method to overcome the computation expense that made other dimensionality reduction techniques infeasible when handling high-dimensional data sets. After incorporating randomized projection into their tool, the authors gained an additional 5% increase in classification and topic separation than in previous methods used. [31]

The following efforts [38, 39, 47] use randomized projection in conjunction with latent semantic indexing. Papadimitriou et al. [47] looking at another information retrieval technique shows positive results in using randomized projections as a pre-processor to the computationally expensive Latent Semantic Indexing. By simply applying randomized projection to their data before computing the Latent Semantic Indexing, their asymptotic running time for the overall system improved from $O(mnc)$ to $O(m(log^2 n + clogn))$, where m and n are the matrix size, c is the average number of terms per document. [47]

Kurimo [38] again showed the promise of using randomized projections as a pre-processor to Latent Semantic Processing. By using the randomized projection techniques the authors were able to "quickly generate approximately orthogonal vectors" [38] of their features, in this case words. Because of the randomized projection method as their approximation, it allowed them to "feasibly use a very large vocabulary." [38] Kurimo's results showed that using the randomized projection technique was "much faster and there were much less complexity problems as the corpus increased." [38]

2.5 Summary

The currently published research efforts into detecting rogue applications have promise. This promise comes in the form of providing higher detection rates than current accepted commercial solutions but as with most any method there are shortcomings to their effectiveness.

Current research into rogue application detection has involved using both static and dynamic techniques. The major efforts using static techniques that also use methodologies from information retrieval and data mining attack the 'curse of dimensionality' by use of feature selection techniques for dimensionality reduction. Though somewhat successful, there is still room for improvement. Feature selection reduces the dimension of the original data set by selecting a subset of the original features mostly based on mutual information algorithms similar to that used by Kolter [37].

There have been no major published efforts that are looking at the concepts of feature extraction for dimensionality reduction when applying methodologies and techniques from information retrieval and data mining. Feature extraction transforms, either linearly or non-linearly, the original feature set into a reduced set that retains the most important predictive information. Feature extraction takes into account predictive information from all of the original features in the data set, not a subset of the features as feature selection does.

The method that has been developed and is presented in this dissertation helps to fill this gap by applying the methods and techniques of randomized projection as a feature extraction technique to my data set. This extraction provides a reduced data set that allows

my prediction algorithms to have more predictive power when compared to the data sets produced by feature selection techniques of mutual information.

CHAPTER 3

RESEARCH EXPERIMENT PLAN AND APPROACH

In this dissertation, I have developed a research effort that applies and extends the information retrieval techniques of *n*-gram analysis and document similarity and the data mining techniques of dimensionality reduction and attribute extraction. This unique combination of techniques, as shown through experimentation, generates a more effective Trojan horse, rogue application detection capability tool suite, whether the application is standalone or embedded within other applications. It will be shown in Chapter 4 that this new tool suite is more effective than current feature selection methods. This suite includes methods for feature/attribute extraction from target applications, sophisticated feature set reduction from the data mining community, and classification capabilities using document similarity from the information retrieval community. The remainder of this chapter provides an introduction and overview of the techniques and methods that were used in my research effort and a description of how the overall experimental plan for the software suite was conducted.

3.1 Technique and Method Introduction

This research effort has developed a unique combination of techniques and methods that have not been presented in this manner before to attach the rogue application

detection problem. Presented below are details regarding all aspect of the tool suite components and functionality.

3.1.1 Feature Definition

In this portion of the research I have chosen a vector space model to represent the data (applications). More specifically, the data is represented as a vector with each dimension being defined as a feature which may or may not have a weight associated with it. [57] Each of the features becomes "an independent dimension in a very high dimensional space." [61] To create these feature vectors, first a definition of what the features contain is needed. For this research effort I used *n*-grams for the features. An *n*-gram is "any substring of length *n*." [8] Here the gram, which is the composite of the substring, is a hexadecimal byte in the ASCII format. The *n*-grams were obtained by first converting each target application in the data set into hexadecimal bytes and then concatenating each set of *n* hexadecimal bytes. These sets of *n* hexadecimal bytes, or features, are defined by a window of size *n* that slides across the converted target application.

3.1.2 Feature Weightings

I experimented with two different types of feature weighting models in this research. First, I used one of the most common vector space weighting models called term-frequency inverse-document-frequency (TFIDF). [63] The TFIDF vector space weighting model "is composed of the product of a *term frequency* and the *inverse document fre-*

quency for each term that appears in the document." [58] The following are the formulas for TFIDF.

$$tfidf_{i,j} = tf_{i,j} \times idf_i \tag{3.1}$$

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}} \tag{3.2}$$

$$idf_i = log\frac{|D|}{|\{d_j : t_i \in d_j\}|} \tag{3.3}$$

In the term frequency formula above (3.2), $n_{i,j}$ is the number of occurrences of the feature in question in application d_j and the denominator is the sum of the number of occurrences of all terms in the application d_j. In the inverse document frequency formula above (3.3), the numerator is the total number of applications in the data set and the denominator is the number of applications where a particular feature t_i appears.

Term frequency, as the name suggests, is the number of times a particular term appears in a document, or in this case, the number of times a particular *n*-gram appears in an application. This can be noted as a measure of importance that the particular *n*-gram has in the application. [58] The inverse document frequency is used to normalize the more common terms. It is defined as the number of documents in which the term appears or in this case the number of applications where the *n*-gram appears. [58] Therefore the product of these two values will provide the "measure of the importance of the *n*-gram in the application and the corpus as a whole." [58] The second feature weighting model that

I used in the research experimentation was a binary model where I assigned a 1 for each n-gram feature that is in the application or a 0 for each feature that is not in the application. This method provides the simplest of weighting methods but is surprisingly powerful.

3.1.3 Dimensionality Reduction

Clearly, using the total number of unique n-grams extracted from all of the applications in the data set to create the feature vectors will present a high-dimensional data problem. To combat this problem, the detection module concentrated on the feature extraction, dimensionality reduction technique of randomized projection. In randomized projection, "the original high-dimensional data is projected onto a lower-dimensional subspace using a random matrix whose columns have unit lengths." [12] This type of projection attempts to retain the maximum amount of information embedded in the original feature set while substantially reducing the number of features required. This feature reduction allows for greater amounts of analysis to be performed. The core concept has been developed out of the Johnson-Lindenstrauss lemma [30] which states that any set of n points in a Euclidean space can be mapped to $\mathbb{R}^t$ where $t = O(\frac{logn}{\varepsilon^2})$ with distortion $\leq 1+\varepsilon$ in the distances. Such a mapping may be found in random polynomial time. A proof of this lemma can be found in [20]. As a comparison approach I used a feature selection method of mutual information as described in Section 2.1.4. Note that this feature selection method for dimensionality reduction is only defined using a binary feature weighting method.

3.1.4 Prediction Methods

The last major step in this research process involves using the vectors that have been created to produce a viable prediction model. For this research the prediction method that I chose was cosine similarity which comes from the information retrieval community.

3.1.4.1 Cosine Similarity

To determine the relevance value between a particular document and a query, measurements are taken of the similarity coefficient between the two vectors. For this research the document was any vector that was in the data set and the query was a particular vector of interest in the data set. This similarity coefficient measure "reflects the degree of similarity in the corresponding terms and term weights" [57] and can be defined as the angle between the two vectors or the dot product between the two vectors as well as others. [61] For this research the cosine similarity function has been used.

Cosine similarity determines the similarity between two data vectors by measuring the angular distance between them. "Cosine has the nice property that it is 1.0 for identical vectors and 0.0 for orthogonal vectors." [61] The following is the formula that was used in this research for computing cosine similarity:

$$CosineSimilarity(Q,D) = \frac{\sum_i w_{Q,i} w_{D,i}}{\sqrt{\sum_i w_{Q,i}^2}\sqrt{\sum_i w_{D,i}^2}} \tag{3.4}$$

This formula computes the similarity between a query Q and a document D. It does so by summing all of the features of the two entities defined in the formula as w. Since I defined

the features of this research to be n-grams, $w_{Q,i}$ is the weight of the ith n-gram in the query and $w_{D,i}$ is the weight of the ith n-gram in the document.

By following a standard information retrieval methodology, applications, in machine readable format, are regarded as documents in the corpus. These "documents" may or may not have a known rogue functionality. The query is an application, again in machine readable format, which may or may not contain a certain type of rogue functionality. This methodology provides an ability to search the corpus with a query and retrieve/identify/predict potentially new rogue applications. This also provides an ability to discover instances of other applications that contain the same type or family of intent, for example one key-logging Trojan used to find another. Below is a small example of how this detection will work.

Corpus

Document 1: `54 68 69 73 20 69 73 20 61 20 74 65 73 74`
Document 2: `54 68 69 73 20 69 73 20 61 20 74 72 61 69 6E`
Document 3: `54 68 61 74 20 69 73 20 61 20 67 72 65 61 74 20 69 64 65 61`

Query

`54 68 69 73 20 69 73 20 6F 6E 6C 79 20 61 20 74 65 73 74`

Using an n-gram size of 4 the similarity values between the query and the three documents are 0.90, 0.58 and 0.08 respectively. For further detail, look at the similarity calculation between the second document in the corpus and the query. Of the total number of n-grams extracted, there were six that were in common between these two documents; `54687973`, `68697320`, `69732069`, `73206973`, `20697320` and `20612074`. Since the numera-

tor of the equation (3.4) looks at the common features between the two documents and the population values of each of these are 1, the resulting numerator value is 6. The denominator looks at all of the features of the two documents and in this case is equal to 10.38. These values give the final similarity calculation of 6/10.38 = 0.58.

3.2 Experimental Plan

In this dissertation research I have applied and extended the information retrieval techniques of *n*-gram analysis and document similarity and the data mining techniques of dimensionality reduction and attribute extraction. The overall hypothesis of this research was that rogue application detection can be accomplished by cleverly combining information retrieval and data mining techniques. The following is the experimental plan that was used for my rogue application detection tool suite. The experiments were divided based on their applicability to my hypotheses defined in Section 1.2 above. The hypotheses are restated here for convenience.

Hypothesis 1:

> *By applying and extending the information retrieval technique of n-gram analysis and the data mining technique of attribute extraction, a Trojan horse, rogue application detection capability tool can be developed that will provide a true positive rate above 98% and false positive rate below 3%.*

Hypothesis 1 Experimentation:

Feature Definition

Following the introduction above, Section 3.1.1, I used a vector space model and represented the features as *n*-grams. To determine the most appropriate values of *n* I performed the following experiments:

1. *n*-grams size of 3
2. *n*-grams size of 4
3. *n*-grams size of 5
4. *n*-grams size of 6
5. *n*-grams size of 7

In each of the above experiments the input was the raw software applications that were in the data set, converted to hexadecimal form. The results, or output, contained the raw feature vectors that were then further processed by the tool suite.

For example, if my application contained the following sequence `43 57 FE DB 7A C3` and the *n* value was 4, then the 'sliding window' would produce the following *n*-grams `4357FEDB`, `57FEDB7A` and `FEDB7AC3`.

Feature Weighting

For this portion of the research I performed experiments with the following different feature weighting models:

1. Term-frequency inverse-document-frequency (TFIDF)
2. Binary weighting - 1 for each *n*-gram feature that was in the application or a 0 for each feature that was not in the application

For the feature weighting aspect of these experiments the inputs consisted of the feature vectors that were created in the feature definition portion of the experiment. The results of this section further refined my data set providing for a more accurate rogue application prediction.

Hypothesis 2:

> *By combining information retrieval and data mining techniques in concert with the dimensionality reduction technique of randomized projection, a more*

accurate solution in detecting rogue attacks can be developed when compared to seminal efforts currently used in research today. A more accurate solution is defined in terms of a 3% increase in overall accuracy when comparing this method to that of the mutual information dimensionality reduction method.

Hypothesis 2 Experimentation:

Dimensionality Reduction

To combat the high-dimensional data problem this research performed investigations using two different methods of randomized projection.

1. Creating a specially defined random matrix and projecting it upon the original high-dimensional data set. This method involved the population of a random matrix which was accomplished in the following two ways:
   a. By selecting vectors that were normally distributed, random variables with a mean of 0.0 and a standard deviation of 1.0.
   b. By selecting the values of 0, +1 or -1 following a probability distribution of 2/3, 1/6 and 1/6 respectively [12].
2. Using a randomly selected number of feature sets and projecting them onto my data set. This method involved experimenting with the following selection of random sets:
   a. Random set size selection of 500.
   b. Random set size selection of 1000.
   c. Random set size selection of 1500.

The inputs for this portion of experiments were the results of the feature definition and feature weighting section. Those high-dimensional feature vectors were reduced to a low-dimensional embedding by my randomized projection algorithms. The goal was to test whether, for my particular data set type, the random matrix or the random set implementation provided a better solution for my rogue application detection capability. The results of these low-dimensional embeddings were the input to my prediction algorithms which determined if a particular application had rogue capability.

3.2.1 Prediction Method

To determine the applicability and correctness of the two hypotheses of this dissertation, this research applied the information retrieval prediction methods of cosine similarity to the low-dimensional embedded feature space. By applying this prediction method consistently across all of the random projection methods introduced above as well as the control method of mutual information, I was able to make a valid and accurate comparison and show that the hypotheses were valid.

Cosine Similarity

I applied the cosine similarity equation (Eq. 3.4) to each query-document feature vector pair in the data set. To determine the correctness I evaluated the results by setting the prediction threshold to 21 separate values starting at 0.0 and continuing in increments of 0.05 until a value of 1.0 was reached.

3.3 Validation

As with any new method, technique or technology that is introduced, a system for determining its accuracy or validity must also be presented. Validation is a key component to providing feasible confidence that any new method is effective at reaching a viable solution, in this case a viable solution to the rogue application detection problem. Validation is not only comparing the results to what the expected result should be, but it is also comparing the results of my techniques and methodologies to other published methods.

The first line of validation used with this research effort was to compare the results/predictions of the rogue applications analysis and detection software suite to the ex-

pected results. The expected results in these experiments were known because the data set was a labeled data set. The second line of validation was to compare these results to those of other published research. To that end, several performance values were used to measure and compare the performance of the experiments conducted in this research effort to those of others published in the literature. These values include true positive rate (TPR), false positive rate (FPR), accuracy and precision. TPR, also known as recall, "is the proportion of relevant applications retrieved, measured by the ratio of the number of relevant retrieved applications to the total number of relevant applications in the data set." [56] In other words TPR is the ratio of actual positive instances that were correctly identified. FPR is the ratio of negative instances that were incorrectly identified. Accuracy is the ratio of the number of positive instances, either true positive or false positive, that were correct. "Precision is the proportion of retrieved applications that are relevant, measured by the ratio of the number of relevant retrieved applications to the total number of retrieved applications," [56] or the ratio of predicted true positive instances that were identified correctly. All of these values are derived from information provided from the truth table. A truth table, also known as a confusion matrix, provides the actual and predicted classifications from the predictor. The following are the mathematical definitions of the performance formulas as well as the truth table (Table 3.1) where *a* (true positive) is the number of rogue applications in the data set that were classified as rogue applications, *b* (false positive) is the number of benign applications in the data set that were classified as rogue applications, *c* (false negative) is the number of rogue applications in the data set

that were classified as benign applications, and d (true negative) is the number of benign applications in the data set that were classified as benign applications. [59]

Table 3.1

Definition of Truth Table

		Actual	
		Positive	**Negative**
Predicted	**Positive**	A	B
	Negative	C	D

Below are the formulas for the four performance calculations that were used in this research effort for validation of the predicted results.

$$TPR = \frac{a}{a+c} \tag{3.5}$$

$$FPR = \frac{b}{b+d} \tag{3.6}$$

$$Accuracy = \frac{a+d}{a+b+c+d} \tag{3.7}$$

$$Precision = \frac{a}{a+b} \tag{3.8}$$

Using these calculated performance values I can validate this work as compared to previously published work. For this I concentrated on published performance value information from Kolter's mutual information. Through these comparisons I was able to substan-

tiate the claim that the methods and techniques described in this research for providing a rogue application detection capability performed “better” than current published methods. Better is defined in terms of absolute comparison of the validation methods presented above.

CHAPTER 4

RESEARCH RESULTS

Using the methods and techniques that were described in Chapter 3 above, I created a functional rogue application detection software tool suite. This tool suite was written in the C programming language with some administrative scripts that were written in Perl. The preliminary experiments were performed on a server class machine running a Linux operating system. All of the experiments using the completed tool suite were performed on commodity Gateway PC's running a Fedora Linux operating system. A proof-of-concept of this tool suite was used to produce the results presented in Section 4.1 Preliminary Results. The completed robust version of the tool suite was used for the results presented in Section 4.2.

4.1 Preliminary Results

These preliminary results were obtained from the proof-of-concept rogue application detection software tool suite. The data used for this preliminary experiment consisted of 267 Windows formatted binary executable files that were obtained from a Windows XP operating system. These files ranged in size from 50KB to 500KB. Integrated within the corpus were 24 files that had been infected with rogue code using the F.B.I. (Finding, Binding and Infecting) binder and six standalone rogue applications for a total of 30 rogue

applications. The Windows applications infected for this experiment were Microsoft Calculator, MS-DOS Command Prompt, Microsoft Notepad and Microsoft 3D Pinball for Windows. The rogue applications used were the CDKey Harvester v0.9, Fearless KeySpy v2.0, LttLogger v2.0, HermanAgent v1.0, ProAgent v2.0 and Recon v2.0. Each docile application was infected with each of the rogue applications using the F.B.I. binder to create 24 infected files. The binder and all rogue applications are freely available for download from the following website, http://www.trojanfrance.com. Table 4.1 contains short descriptions of the rogue applications used in this experiment.

Table 4.1

Descriptions of Rogue Applications

CDKey Harvester v0.9	Searches victim's registry for Online Game CD Keys and sends them to the attacker through email
Fearless KeySpy v2.0	keystroke logger
LttLogger v2.0	keystroke logger that can completely remove itself at a specified time or after a specific amount of collection
HermanAgent v1.0	password stealer where information is passed back to the attacker through email
ProAgent v2.0	monitoring and surveillance tool that captures data from webcams, screenshots and microphone usage
Recon v2.0	keystroke logger that can disable anti-virus and firewall software

For this proof of concept, I limited the size of the *n*-grams to a 4-byte window. This provided a boundary of the potential size of a given feature vector. Using the predefined *n*-gram size of 4 there are approximately 4×10^9 number of total possible *n*-grams.

The data set described above contained 6,997,927 unique *n*-grams. For the dimensionality reduction stage I used a random matrix and projected it upon the original high-dimensional data set to produce a new low-dimensional embedding that contained 500 features. The random matrix for the projection was created by randomly selecting values to populate the vectors of the matrix that were normally distributed random variables with a mean of 0.0 and a standard deviation of 1.0. As a comparison and validation for the randomized projection approach I implemented the feature selection mutual information approach similar to the one described in [37]. This separate data set contains 500 features identified by the mutual information algorithm of the original data set. I applied the cosine similarity algorithm to these two data sets with the following positive results presented in Tables 4.2, 4.3 and 4.4.

Table 4.2

Performance Values Comparing Reduction Methods for Threshold of 0.55

Performance Metric	**Random Projection**	**Mutual Information**
TPR	0.8	0.67
FPR	0.05	0
Accuracy	0.93	0.96
Precision	0.6	1

In each case it can be seen that the randomized projection method provided a higher TPR than the comparison Mutual Information method. This means that in each of the above cases the mutual information method was mis-identifying a larger amount of rogue applications as being benign than the randomized projection method. These results are

Table 4.3

Performance Values Comparing Reduction Methods for Threshold of 0.60

Performance Metric	**Random Projection**	**Mutual Information**
TPR	0.8	0.73
FPR	0	0
Accuracy	0.98	0.97
Precision	1	1

Table 4.4

Performance Values Comparing Reduction Methods for Threshold of 0.65

Performance Metric	**Random Projection**	**Mutual Information**
TPR	0.63	0.6
FPR	0	0
Accuracy	0.96	0.95
Precision	1	1

significant suggesting that this technique can maintain a high precision without sacrificing accuracy or TPR. The methods used in previous research, mentioned in Chapter 2, report accuracy ratings ranging from 93% to 98%, so the results presented here were comparable and in some cases outperformed other methods.

4.2 Experimental Results

For the full version of the experiments, I used the completed rogue application detection tool suite. As stated above, all of these experiments were run on commodity hardware running the Fedora Linux operating system. It is significant that I was able to complete all of these experiments on commodity hardware. It shows that large specialized

machines are not needed to perform rogue application detection and that this work can be broadly applied across almost any level of architecture that researchers/developers may have and still gain the significantly positive results that were obtained and are discussed below. In addition, this software and the methods that it supports can easily take advantage of commodity cluster hardware for substantial gains in performance.

4.2.1 Data Set

The data set that was compiled together for the experiments described in this section consisted of 1544 Windows formatted binary executable files. None of the files in the data set were larger than 950KB. Of these files 303 were extracted from a fresh installation of the Windows XP operating system. Another 406 were extracted from a fresh installation of Windows Vista operating system. Both of these sets were obtained by installing the respective operating system in a virtual environment that was installed on a commodity PC. These virtual environments were not connected to the Internet and therefore provided a safe location. This ensured that it would allow for application extraction without the worry of rogue infiltration during the gathering phase of the research effort. This process provided a total of 709 files that were in the data set and that were considered benign. The remaining 835 files for the data set were rogue Trojan horse applications that were downloaded from various websites on the Internet including http://www.trojanfrance.com and http://vx.netlux.org.

4.2.2 Presentation of Results

The outcome of these experiments has produced a cadre of significant results that not only show success when compared to the research hypotheses but will also benefit the rogue application detection community. This successful research provides an additional pathway for attacking and effectively detecting rogue applications using static methods. As stated in Section 2.3 above static methods offer many advantages over their dynamic method counterparts, including but not limited to the ability to perform accurate rogue application detection without the danger of actually executing the potential rogue application and risking infecting more machines.

For the remainder of this chapter I will present a selected subset of the results that were obtained using the data set described in Section 4.2.1 above and following the experimental plan presented in Section 3.2. The entire set of results can be found in graphical form in Appendix B of this document. Each graph in Appendix B represents the validation calculations, described in Section 3.3, for each combination of *n*-gram value, dimensionality reduction value and data set instantiation (described below in Section 4.2.2.1). For each graph I present the validation values for each cosine similarity prediction threshold for all seven methods that were examined during the course of this research effort. As a reminder, those methods are the control method of mutual information with the binary feature weightings, the randomized matrix projection using the normally distributed random variables with both the TFIDF and binary feature weightings, the randomized matrix projection using the probability-based random variables with both the TFIDF and binary

feature weightings and the random set projection with both the TFIDF and binary feature weightings.

4.2.2.1 How to Read Graphs

To completely understand how to read these graphs the following example explanation of results for an *n*-gram selection of 3, the data set instantiation of the data portion and a feature reduction of 1000, refers to Figure 4.1 below. The upper left quadrant contains the validation accuracy calculation results for the range of cosine similarity threshold values. For this graph we are looking for the highest peak which will equate to the highest accuracy value for this configuration. In Figure 4.1 we can see that for the control, which is mutual information and is the red line on the graph, the highest accuracy value is 0.93 or 93% accuracy at a threshold value of 0.8. We can also see that each of the random matrix projection methods outperformed this value with the randomized matrix projection using the normally distributed random variables and binary feature weightings, the green line, performing the best at 0.98 or 98% at a threshold value of 0.35. Using these same two methods (mutual information and random matrix projection using normally distributed random variables and binary feature weighting) and respective threshold values (0.8 and 0.35) we can look at the lower left quadrant, true positive rate (TPR), and note that those methods performed identically with a TPR of 0.98 or 98%. Turning our attention to the upper right quadrant, which is the false positive rate (FPR), we see that the randomized projection method (green line) again outperformed the mutual information method (red line) with values of 0.03 or 3% and 0.13 or 13% respectively. Finally,

looking at the lower right quadrant we see that the precision values for the randomized projection method is at 0.97 or 97% and the mutual information method is 0.89 or 89%. Following this same process, all of the results graphs in Appendix B can be interpreted to gain insight into the performance of all of the methods used in this research experiment.

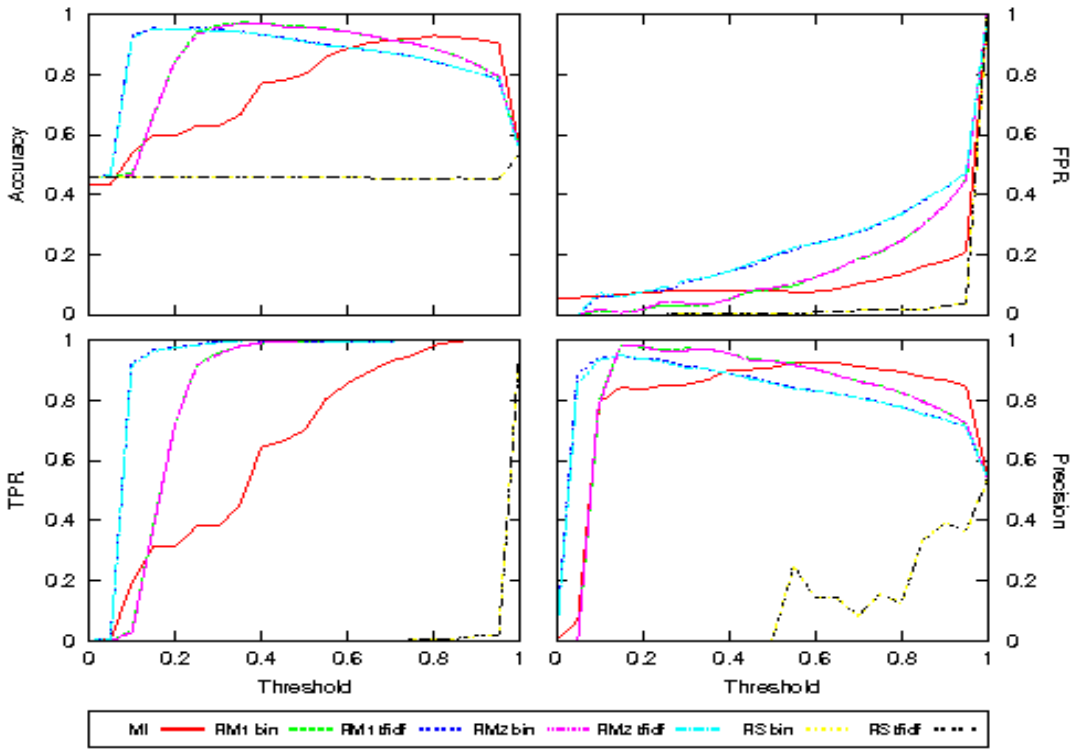

Figure 4.1

3-gram, Data portion data set, 1000-features

We can see that for this particular *n*-gram, feature set size and data set instantiation there is a significant decrease in precision when comparing the mutual information method with the random matrix projection method. This reduction can be attributed to the large difference in the FPR rate. In laymen's terms, both of the example methods we are looking

at provide the analyst with an excellent number of accurately detected rogue applications but the randomized projection method does this with far less incorrectly identified benign applications. This allows the analyst to complete his job much quicker and provides the analyst with additional confidence that the methods are producing precise results.

4.2.2.2 Data Set Instantiation

To assist in obtaining a more accurate answer to the questions posed in the hypotheses, I created multiple instantiations of the data set. The first instantiation involved using the entire or whole application itself. This instantiation of the data is the one that is used by all of the researchers that are mentioned in the literature survey described in Chapter 2. The remaining three were created through extracting and combining well-known defined sections from the whole application. The second and third instantiations were created by extracting the code and data sections from each application using the PE Explorer tool from Heaventools. [1] To confirm the accuracy of this tool I hand dissected several of the applications in the data set, compared my results to the results provided by PE Explorer, and found the tool to be very accurate. To create the fourth instantiation of the data set I combined the data and code sections together. These additional instantiations were done to determine if extracted sections of each application could prove more fruitful in detection than by just using the entire application. The thought process behind creating these multiple instantiations was as follows. Since all of the applications in the data set were valid Windows format executables, there would have to be an inherent similarity in all of them. This comes from both structure and header contents which may hamper

attempts to produce valid and viable rogue application detection. By extracting the data and code sections I was able to remove this inherent similarity and allow the detection methods to concentrate on the true differences in the applications. It must be noted that with the combined data and code data set instantiation a 'false set' of features is created at the point of fusion. This set is an extremely small set, at most for these experiments it will be 7, when compared to the entire set of features that are extracted and therefore will not hamper any detection capabilities of the tool suite.

When the results are examined from a data set instantiation viewpoint holding the remaining variables of *n*-gram size and dimensionality reduction size constant, it is clear that using the extracted data set instantiations provided a considerable increase in accuracy when compared to using the entire or whole application. It can be further derived that the code instantiation provides better results than the data instantiation. Even better results can be obtained by the combination of the data and the code instantiations. An example of this increase in accuracy can be seen in Figures 4.2, 4.3, 4.4 and 4.5. These figures are for the 4-gram experiments where the dimensionality was reduced to 1500 features. For the whole instantiation of the data set (Figure 4.2) at a threshold of 0.9 the mutual information method obtains an accuracy of 0.90 or 90% and at a threshold of 0.2 the random matrix method with a normalized-based matrix and the binary feature weighting obtains an accuracy of 0.97 or 97% accuracy. Compare this to the code instantiation of the data set (Figure 4.4) that has an accuracy of 0.93 or 93% for the mutual information and 0.997 or 99.7% for the random matrix projection. Similar results can be seen in both

the data instantiation (Figure 4.3) and the combination of the data and code instantiation (Figure 4.5).

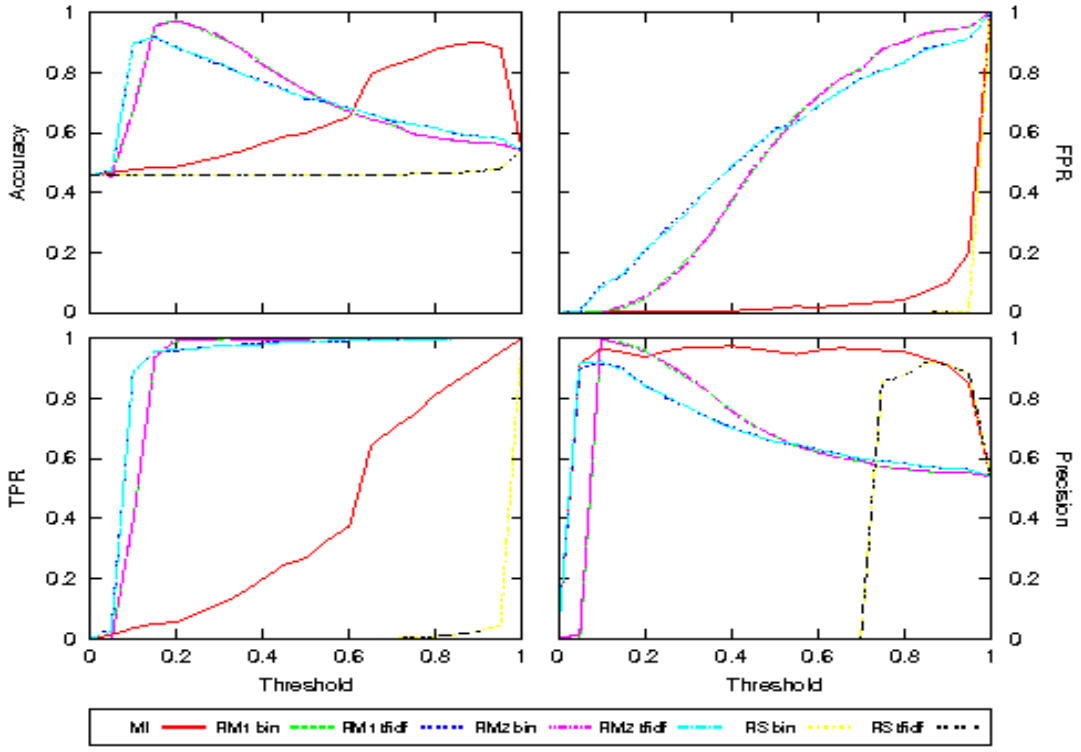

Figure 4.2

4-gram, Whole portion data set, 1500-features

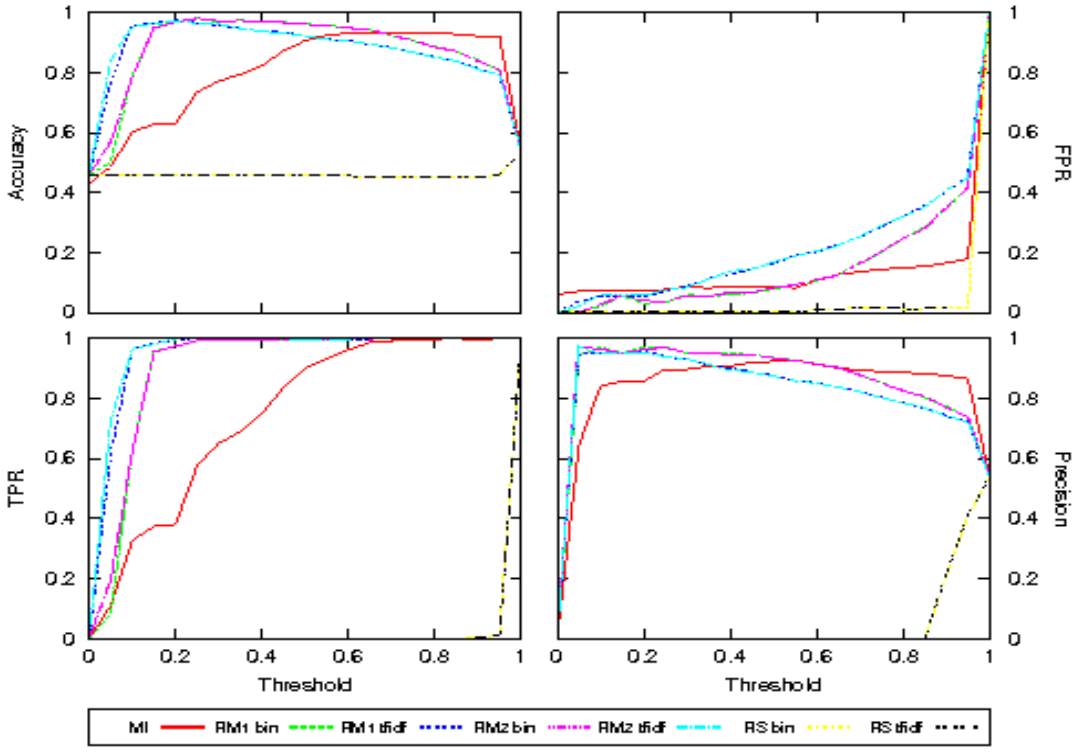

Figure 4.3

4-gram, Data portion data set, 1500-features

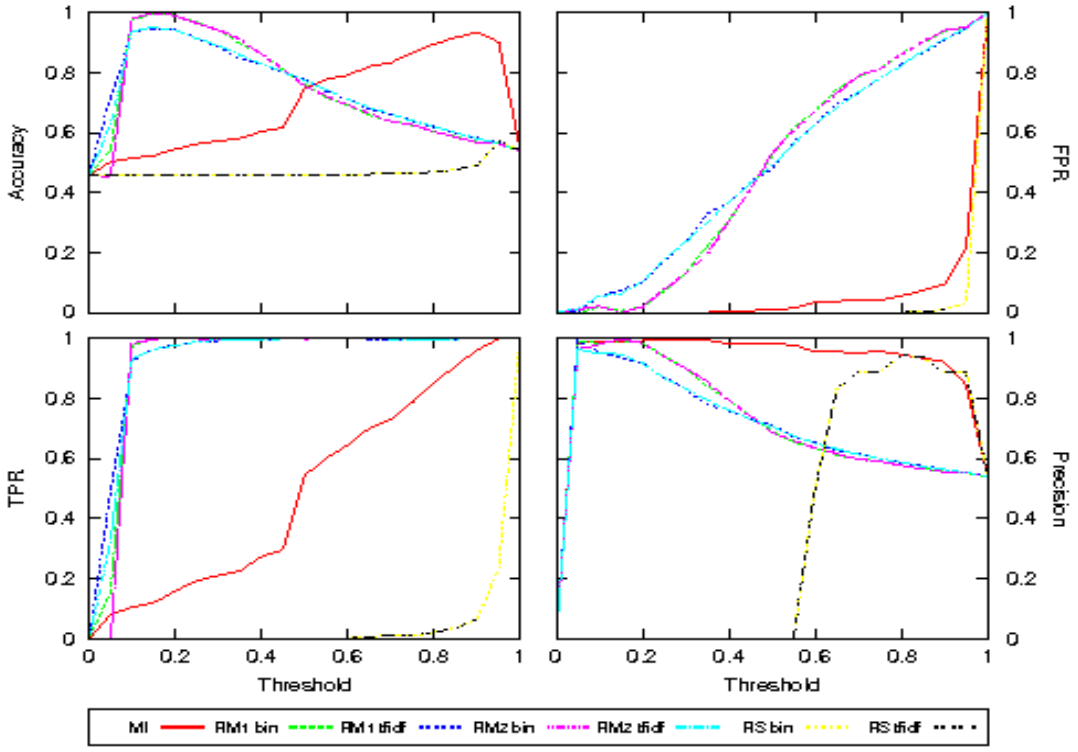

Figure 4.4

4-gram, Code portion data set, 1500-features

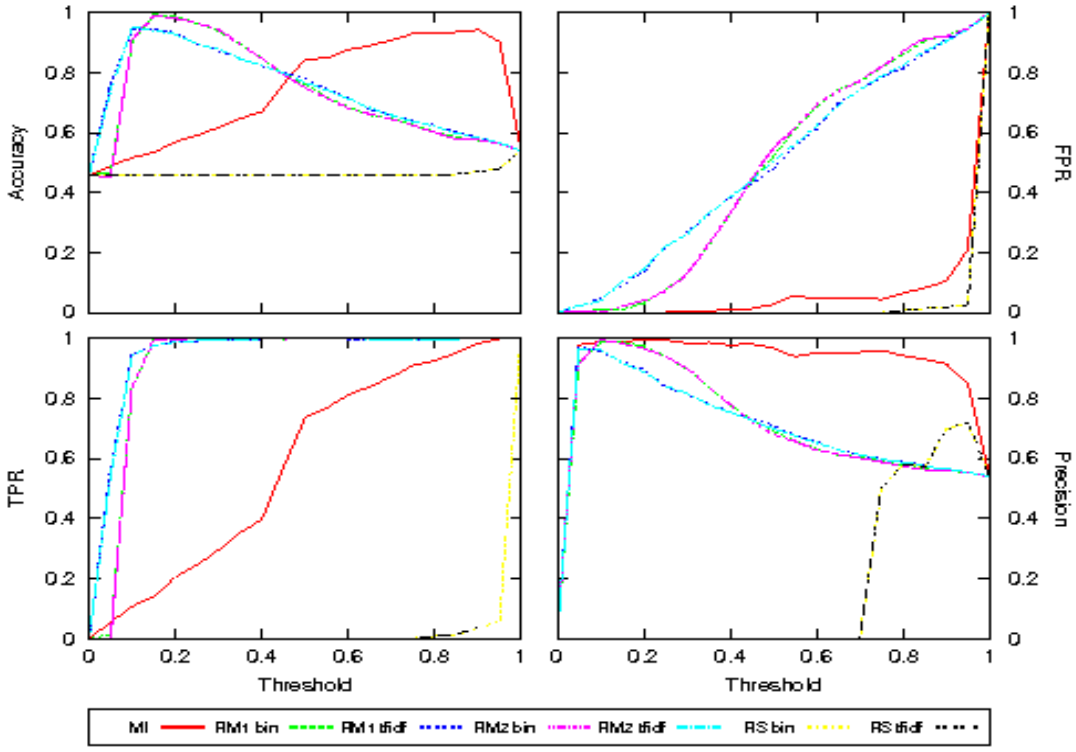

Figure 4.5

4-gram, Combination portion data set, 1500-features

This important result of the extracted instantiations outperforming the whole application can be seen throughout the experiment. This is a positive significant step in that this type of slicing of applications to make a rogue application detection determination has not been published before at this level. By extracting these sections from an application, this makes the data search space much smaller and therefore allows for a faster detection time and a more accurate detection because of the ability to include more applications in the detection corpus.

Again, it can be determined throughout the experiment that by looking at all of the results, choosing the combination of the data and code instantiations performs at the highest level. Staying with the same example as above the combination instantiation outperformed the whole by 5% accuracy, by 1.5% TPR, by 7.4% precision and by 9.4% FPR.

4.2.2.3 *n*-gram Variation

When the accuracy results are examined from an *n*-gram variation viewpoint, holding the remaining variables of data set instantiation and dimensionality reduction size constant, it was not as clear cut as with the data set instantiation but a few significant items surfaced. First, it could be seen that the control method of mutual information performed best at *n*-gram values of 4 and 5. The *n*-gram value of 4 is the value that has been published in the literature as the best expected value for mutual information. It must be noted that there were some anomalies to this statement but for the majority of the experiments the *n*-gram values of 4 and 5 performed the best for the control method of mutual informa-

tion. As for the random projection methods, the best results in terms of highest accuracy values were achieved at the same *n*-gram values of 4 and 5. Using an *n*-gram value of 3 produced significant results as well; however, the results using the *n*-gram values of 4 and 5 were better in most cases. The *n*-gram values of 6 and 7 produced good results for the random projection methods, but in some cases produced poor results for the mutual information method. As for the speed, a smaller choice for the *n*-gram size produced a data search space that was much smaller and therefore allowed for a much faster calculation of the low-dimensional embeddings. Even though there was no clear cut winner on *n*-gram size, there were significant results that can be noted and used in future research experiments.

For an example of these performance differences refer to Figures 4.6, 4.7 and 4.8. In this example, I have held the dimensionality reduction feature size to 1000 and the data set instantiation to the code instantiation. Observing the accuracy values, upper left quadrant, for each of these three graphs it can be seen that for mutual information the best value is recorded in Figure 4.8, 5-gram, with an accuracy of 0.92 or 92% at a threshold of 0.9 with 4-gram next at an accuracy value of 0.88 or 88%. Both of these values are higher that the 3-gram (Figure 4.6) accuracy value of 0.87 or 87%. With respect to the random matrix method with a normalized-based matrix and the binary feature weighting, it obtained the best accuracy value of 0.997 or 99.7% accuracy for the *n*-gram value of 4 (Figure 4.7). An almost identical value of 0.996 or 99.6% was obtained when the *n*-gram value was changed to 5. Both of these values were obtained at a threshold value of 0.15. For this example even the 3-gram accuracy value is significant at 0.994 or 99.4% but still

lower than both the 4 and 5 gram values. It must be noted that the values obtained for the randomized matrix projection are at least 7% higher than those for mutual information, with a maximum increase of 12%. The remaining three quadrants report similar increases in performance. For TPR, lower left quadrant, the randomized matrix projection showed at least a 4% increase over mutual information, with a maximum increase of 20%. At least an 8% increase in precision, lower right quadrant, was shown with a maximum increase of 11%, and for FPR, upper right quadrant, a 9% decrease over mutual information again with a maximum decrease of 13%. These are important results comparing these methods at their best *n*-gram size determined through the experiments. Note that similar results were obtained throughout the experiments and that these are presented here as a representation.

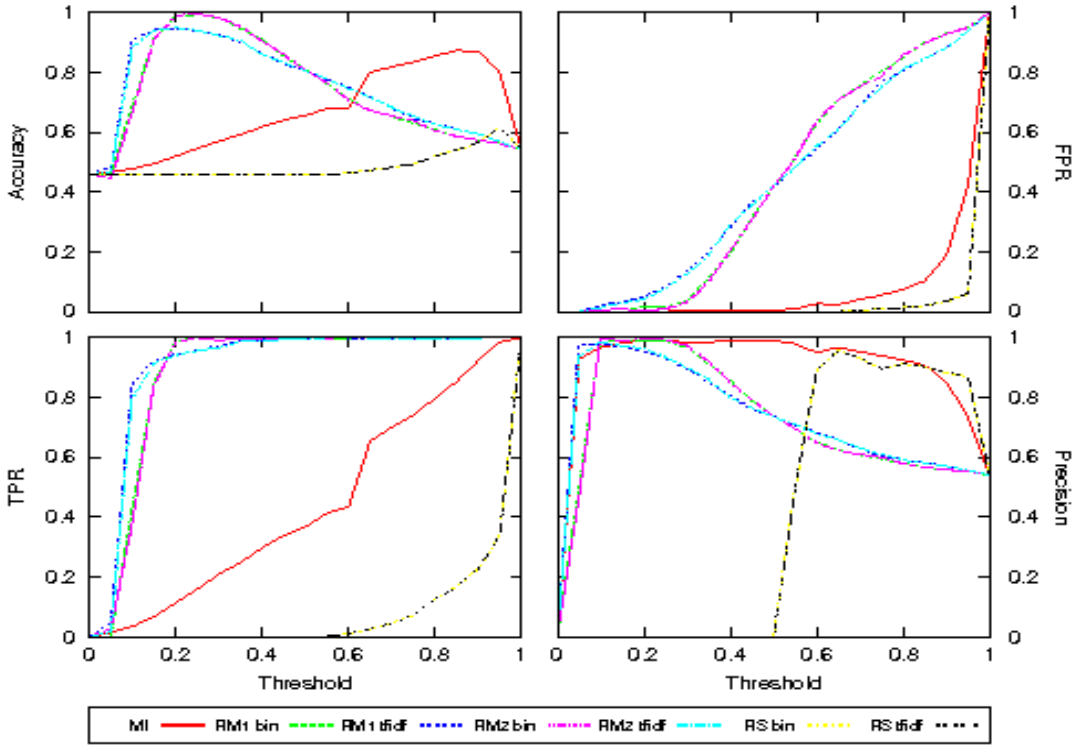

Figure 4.6

3-gram, Code portion data set, 1000-features

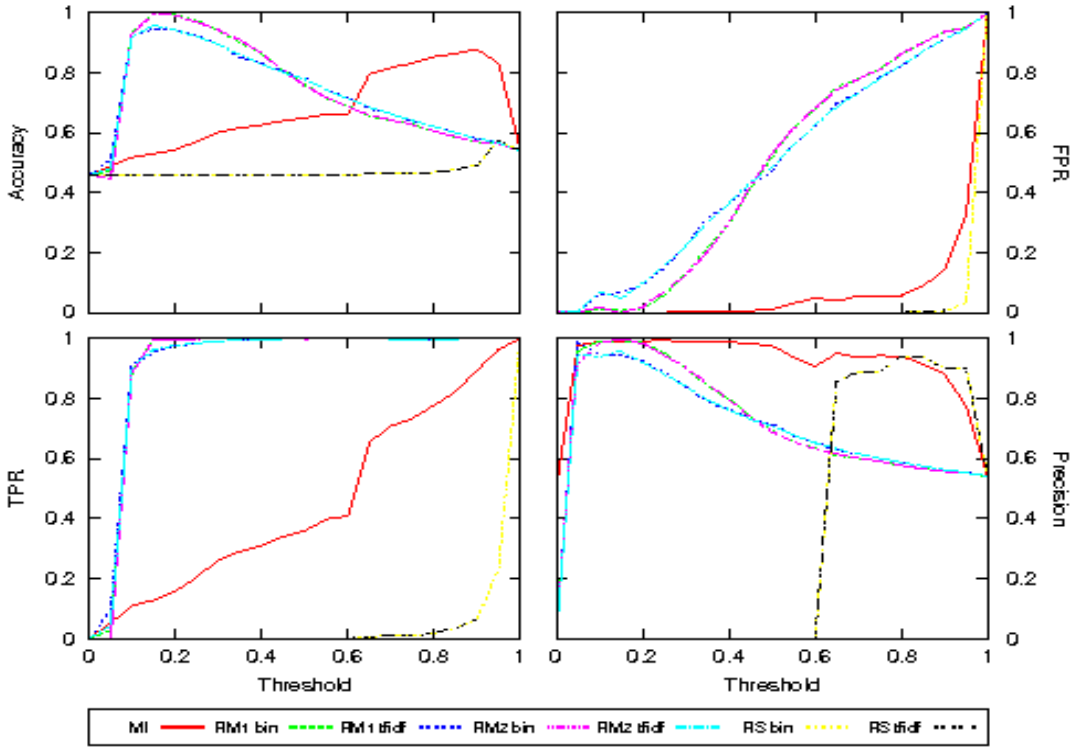

Figure 4.7

4-gram, Code portion data set, 1000-features

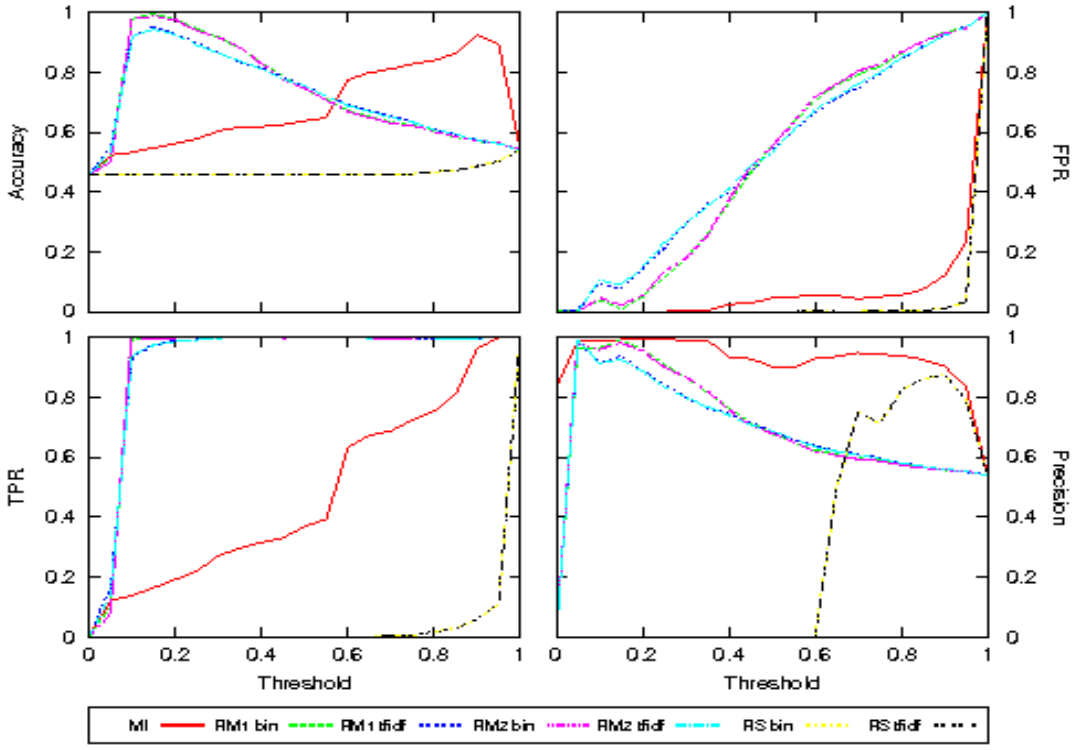

Figure 4.8

5-gram, Code portion data set, 1000-features

4.2.2.4 Dimensionality Reduction Variation

When the results are examined from a dimensionality reduction variation viewpoint holding the remaining variables of data set instantiation and *n*-gram size constant, some positive results were noticed. Of particular note another confirmation of my implementation of the mutual information method was achieved. Logic would say that the higher dimensionality reduction size of 1500 would outperform the other two sizes used in this experiment; however, as the background literature on the mutual information method noted, that was not the case. My experimental results, as well as those presented in the literature, showed that a dimensionality reduction size of 500 performed the best for the mutual information method. The one exception to that was using an *n*-gram size of seven. In this case, the performance across all methods increased as the dimensionality reduction size increased. As for the randomized projection methods, they produced very similar results between the 1000 and 1500 size reductions and a slight decrease in performance for the 500 size reduction. In all of the cases, even at the 500 reduction size, comparing the results of the mutual information method to the randomized projection methods showed that the randomized projection method had a better performance in all categories of performance measurements that were used in this experiment.

An example of the change in performance with respect to dimensionality reduction size can be seen in Figures 4.9, 4.10 and 4.11. For these examples the data set instantiation is set to the combination of the data and code instantiations and the *n*-gram size is held constant at a size of 5. The mutual information method has a decrease of 3% from 0.98 or 98% at a feature reduction size of 500 to 0.95 or 95% at a size of 1500. The random

matrix method with a normalized-based matrix and the binary feature weighting has an increase of 1% to 0.99 or 99% when moving from a feature reduction size of 500 to 1500. Similar performance degradations can be seen in the remaining performance measures for the mutual information method: decrease in TPR by 2%, increase in FPR by 4% and decrease in precision by 3%. As for the random matrix projection, similar performance gains can be seen in both FPR, 2% decrease, and precision, 2% increase. The TPR value stayed constant through the dimensionality reduction changes. Again, as in all of the above variations, the randomized projection methods outperformed the mutual information method. This again shows the significance of this method when compared to the standard accepted method of mutual information.

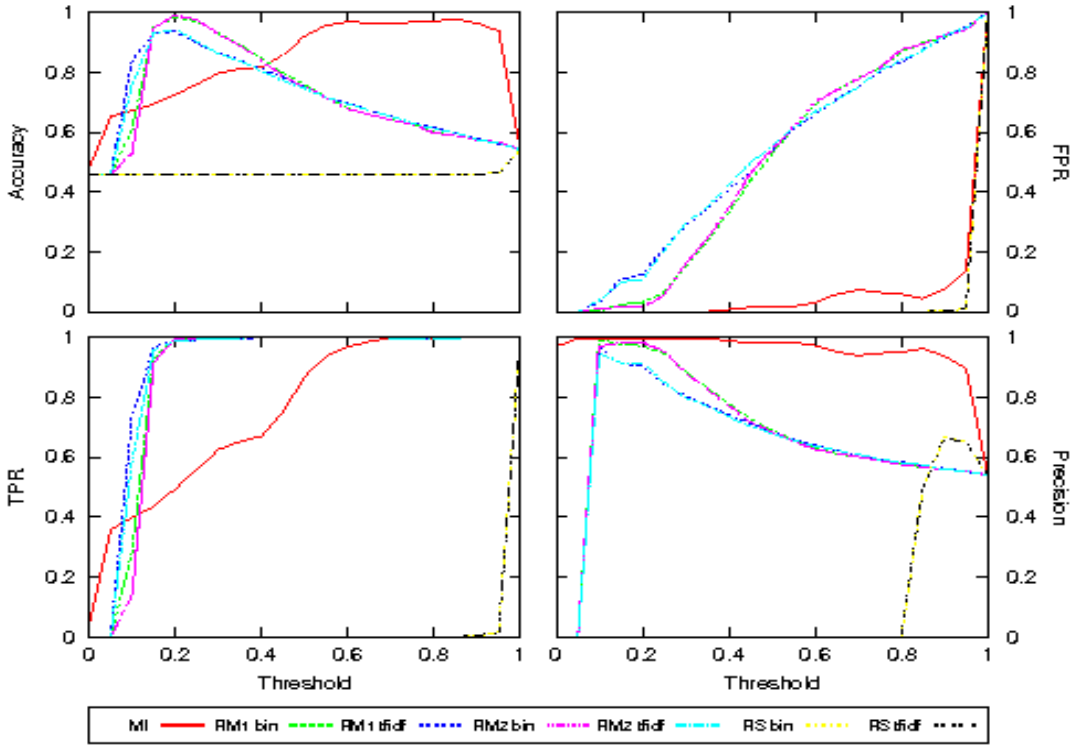

Figure 4.9

5-gram, Combination portion data set, 500-features

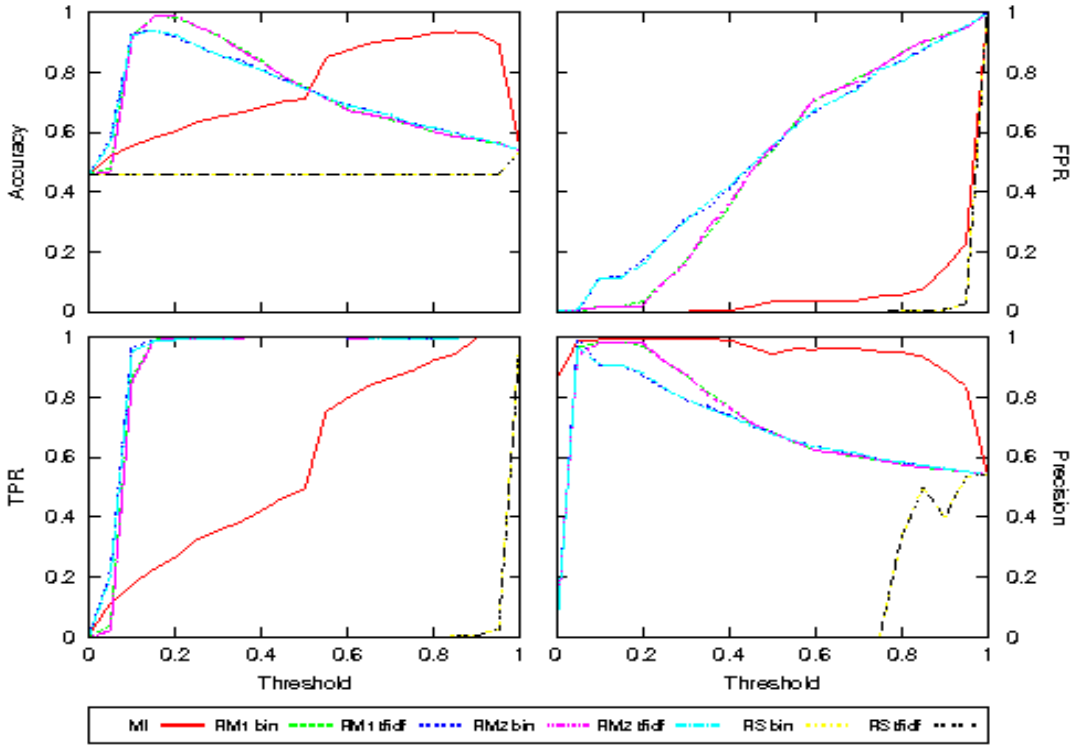

Figure 4.10

5-gram, Combination portion data set, 1000-features

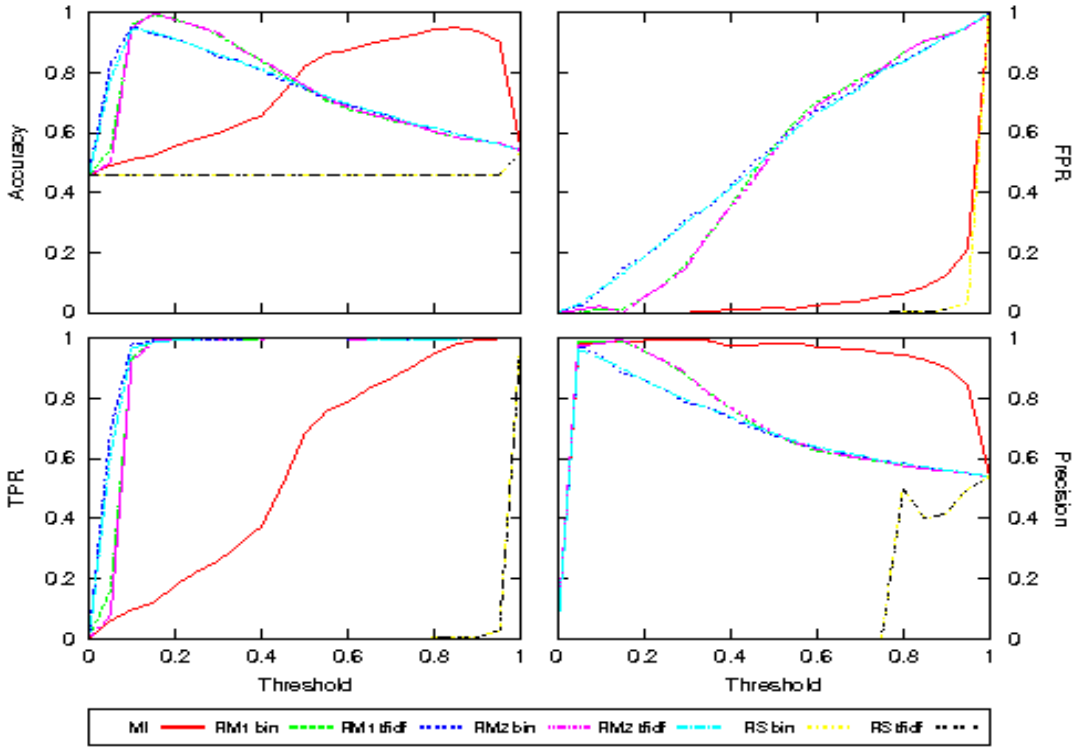

Figure 4.11

5-gram, Combination portion data set, 1500-features

4.2.2.5 Hypotheses Answered

The results provided with this research effort show that it is possible to create a viable rogue application detection capability using the concepts of randomized projections. As can be seen in the last several subsections, applying my randomized projection methods as a dimensionality reduction technique has provided several significant and positive results. This can be noted with any of the three dissections that were presented. In this section, I will provide additional examples that were obtained from this effort to show as proof that the hypotheses defined for this research have been answered in the affirmative. Before presenting these results I will restate each of the hypotheses for the reader's convenience.

Hypothesis 1:

> *By applying and extending the information retrieval technique of n-gram analysis and the data mining technique of attribute extraction, a Trojan horse, rogue application detection capability tool can be developed that will provide a true positive rate above 98% and false positive rate below 3%.*

To answer Hypothesis 1 I was able to show conclusively that a true positive rate greater then 98% and a false positive rate less than 3% can be obtained. One of the many examples of this can be seen in Figure 4.12. In this particular example I have restricted the data set instantiation to the code instantiation, the *n*-gram size is restricted to 6 and the dimensionality reduction size is restricted to 1000 features. For each of the randomized matrix projection methods in this example I obtained greater than a 98% true positive rate performance value and a less than 3% false positive rate. In fact, for the entire set of research experiments which consisted of 60 separate experiments for each of the seven

dimensionality reduction methods, the random matrix method with a normalized-based matrix and the binary feature weighting reached this goal 52 times. The random matrix method with a probability-based matrix and the binary feature weighting also produced a result of 52 times reaching or surpassing the goal. The random matrix projection using the TFIDF feature weighting did not fair as well but still had an overall average of 0.97 or 97% for both the probability-based and the normalized-based random matrices. Note that all of the random matrix TPR performance averages were between 2% - 4% higher than the mutual information results. Again, this is significant showing the viability of these methods.

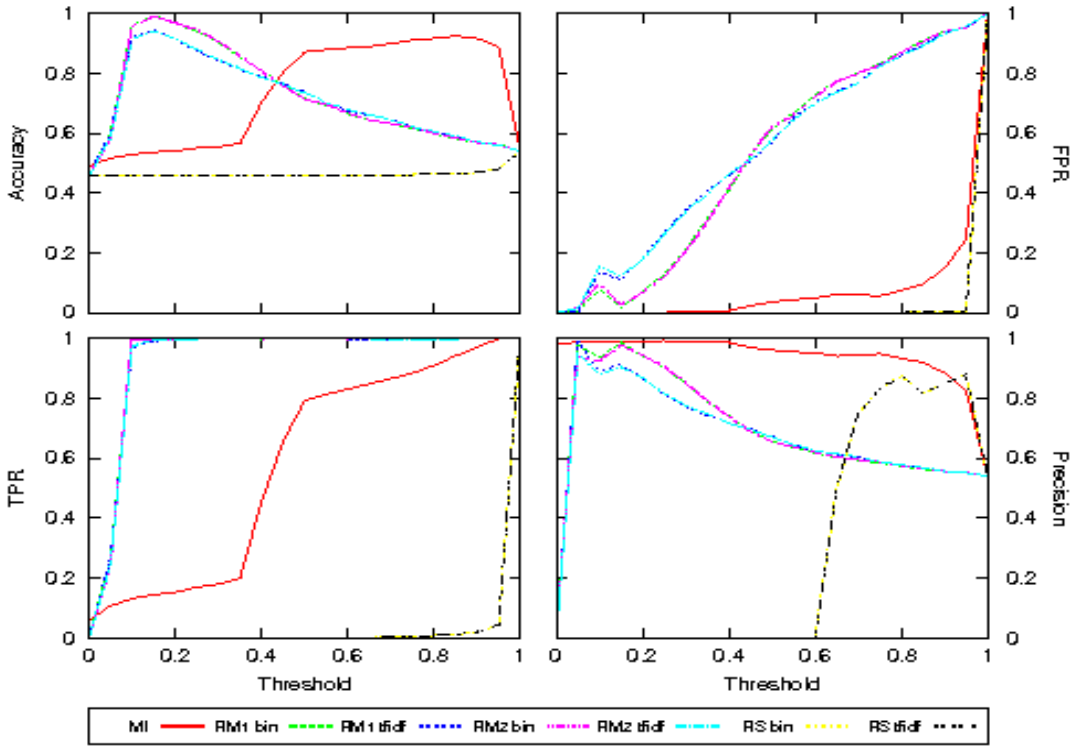

Figure 4.12

6-gram, Code portion data set, 1000-features

With regards to the FPR rate and the entire set of research experiments, these were not as impressive but still very positive. The random matrix method with a normalized-based matrix and the binary feature weighting reached this goal 33 times. More importantly for the code instantiation of the data set this method reached the goal of 3% or less FPR on all 15 experiments. The random matrix method with a probability-based matrix and the binary feature weighting also produced a result of 31 times reaching or surpassing the goal. Here again for the code instantiation of the data set the method performed extremely well reaching or surpassing the goal 14 of the 15 experiments. As for the TPR results the TFIDF weighting experiments did not perform as well as the binary weighting experiments; however, in a good number of cases, they outperformed the mutual information experiments, in some cases by as much as 8%.

These research results are significant proving that a rogue application detection capability tool can be developed within strict performance bounds using information retrieval and data mining techniques. These TPR and FPR results show that the feature extraction method of randomized projection using cosine similarity as a prediction method also outperforms the accepted feature selection method of mutual information at almost every instance of the experiment and at some instances by a great margin. For example using the whole portion of the data set, an *n*-gram size of 4 and a dimensionality reduction size of 1000 features, the random matrix method with a normalized-based matrix and the binary feature weighting had a 45% higher TPR value than the mutual information method using the binary feature weighting. In another example, this time using the code instantiation of the data set, an *n*-gram size of 6 and a dimensionality size of 1500 features, the

random matrix method with a normalized-based matrix and the binary feature weighting had a 22% lower FPR value than the mutual information method using the binary feature weighting. Again these are just a few examples of the entire experimental set but they show the viability and value of using the random projection method in developing a rogue application detection tool suite.

Hypothesis 2:

> *By combining information retrieval and data mining techniques in concert with the dimensionality reduction technique of randomized projection, a more accurate solution in detecting rogue attacks can be developed when compared to seminal efforts currently used in research today. A more accurate solution is defined in terms of a 3% increase in overall accuracy when comparing this method to that of the mutual information dimensionality reduction method.*

To answer Hypothesis 2, I was able to show conclusively that when compared to the control method of mutual information an overall accuracy gain of 3% can be obtained. One of the many examples of this can be seen in Figure 4.13. In this particular example I have restricted the data set instantiation to the data section of the applications, the *n*-gram size is restricted to 7 and the dimensionality reduction size is restricted to 1500 features. For each of the randomized matrix projection methods in this example I obtained an increase of greater than 3% in the overall accuracy performance value when compared to the control method of mutual information. In fact, for the entire set of research experiments which consisted of 60 separate experiments for each of the seven dimensionality reduction methods, the random matrix method with a normalized-based matrix and the binary feature weighting reached this goal 42 times. The random matrix method with a probability-based matrix and the binary feature weighting produced a result of 39 times

reaching or surpassing the goal. The random matrix projection using the TFIDF feature weighting did not fair as well, but still had an overall average that produced an accuracy rate that was greater than the mutual information method for both the probability-based and the normalized-based random matrices. On average, the randomized matrix projection method when using the binary feature weighting produced an overall difference of over a 5% higher accuracy value compared with that of the mutual information method. This is again significant, showing the viability of these methods.

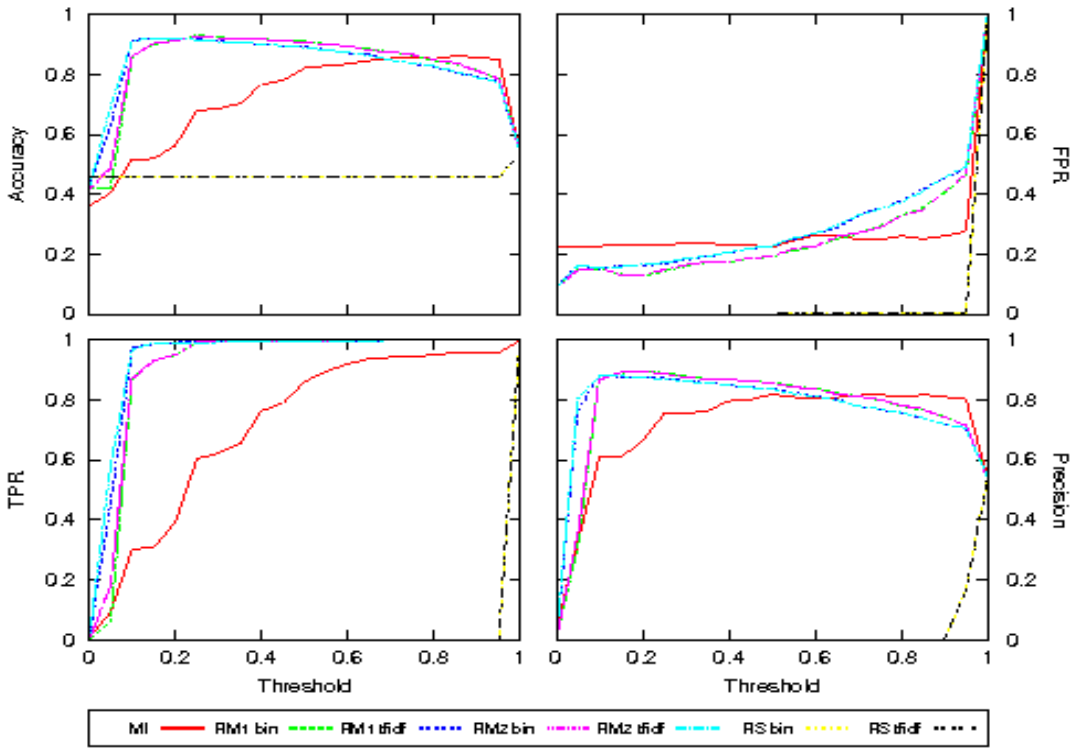

Figure 4.13

7-gram, Data portion data set, 1500-features

As with the TPR and FPR results described above for Hypothesis 1, these research results are significant proving that by using randomized projection as a feature extraction method a rogue application detection capability tool can be developed that out-performs feature selection methods. These accuracy results again show that the feature extraction method of randomized projection outperforms the accepted feature selection method of mutual information in an overwhelming majority of instances of the experiments and at some instances by a great margin. For example using the whole portion of the data set, an *n*-gram size of 4 and a dimensionality reduction size of 1000 features, the random matrix method with a normalized-based matrix and the binary feature weighting had an overall accuracy performance value difference of over 28% higher than the mutual information method using the binary feature weighting. In another example, this time using the data instantiation of the data set, an *n*-gram size of 7 and a dimensionality size of 1000 features, the random matrix method with a normalized-based matrix and the binary feature weighting had an overall difference of almost 13% higher than the mutual information method using the binary feature weighting. As with the TPR and FPR values, these are just a few examples of the entire experimental set, but they show the viability and value of using the random projection method in developing a rogue application detection tool suite.

4.3 Summary

Although I have presented results that clearly show that the hypothesized method for rogue application detection is viable, not every method performed as well as expected. As can be seen in most every graph in Appendix B, the random set method did not perform

well. I have had good results in the past with this method [7] but for this particular set of experiments this method greatly underperformed. It is my belief that this method may not be suitable to such a high dimension and therefore should only be used when the original feature space is not as large as the one that is presented in these experiments. Another potential reason that the random set feature extraction method did not perform as well as expected could have been because of the prediction method that was used. With another prediction method such as nearest neighbor, this method might have performed just as well as its random matrix projection counterpart. Further experimentation with this method will be something I will continue to look at in the future to determine a better bounds on its applicability.

Another aspect that did not perform as expected was the TFIDF vector space feature weighting method. In almost every experiment the TFIDF method was out-performed by its binary feature weighting counterpart. In contrast to the random set projection discussed above, the prediction algorithm of cosine similarity is traditionally used when TFIDF feature weighting is applied. Therefore, additional prediction algorithms tested, at least on the surface, do not seem to be appropriate. However, I plan to do further experimentations in the future. For now, the conclusion from this research effort is that with this type of non-traditional data TFIDF is not applicable, even though in traditional information retrieval this method has had great success.

These two items aside, the results of this research effort have provided the rogue application detection community with a powerful tool for their toolbox. As stated in the motivation section of this dissertation, it is my belief that a multi-pronged defense ap-

proach where there are several weapons in the administrator's toolbox is needed to attack this problem. By developing a more effective rogue application detection tool through the use of information retrieval and data mining methodologies, as well as dimensionality reduction through randomized projections, I have provided one more weapon in the consumer's toolbox to attack this problem. No one is immune from these malicious attacks; from the corporation to the unsuspecting home user, everyone is at risk. Therefore, an entire range of consumers can benefit from the promising results of this important research effort.

CHAPTER 5

CONCLUSION AND FUTURE PLANS

5.1 Conclusion

The results presented above provided through actual experimentation both support and validate each of the hypotheses stated in Chapter 1 that applying the dimensionality reduction technique of randomized projection before using the cosine similarity algorithm has merit in determining if an application may contain rogue functionality. There is no claim that this is a complete solution, rather a tool designed to fit into the security administrator's toolbox as a data point or first pass to help reduce the number of applications needing review. This potential reduction in number of applications to sort through can provide an administrator or analyst with valuable time saving by not having to analyze applications that clearly do not contain rogue functionality. With more and more applications not being developed "in-house" this is a positive result for those responsible for providing secure solutions.

5.2 Future Work

This section includes ideas of future topics to concentrate on and directions to take the rogue application detection tool suite on now that this portion of the research effort has

been completed. Future work with this research will follow two major pathways: first, continual improvement of the tool suite to include additional manipulations of the data set and expansion of prediction methods, and second not to be limited to the general task at hand of developing a rogue application detection capability but also using the underling methods and techniques by applying them to a diverse set of fields.

5.2.1 Continual Improvement

In the future I am interested in integrating additional prediction algorithms into the tool suite, such as a decision tree, neural network and k nearest neighbor algorithms. Furthermore, with this research using multiple prediction methods there is also an experimental track that involves how to cleverly combine the results of the methods to produce one result that has the potential of being more accurate than any of the individual models by themselves. This will involve applying a voting method where each model will get a vote and the majority vote becomes the prediction. Another method introduced by Todorovski et al. [67] is called Meta Decision Tree (MDT). The leaf nodes of an MDT provide the prediction model to use and the internal decisions are made "based on the class probability distributions for the given [query]." [67] This may provide the mechanism to extend the method of random set projection. Another potential improvement to the tool suite will be to add the capability to start the sliding window outside of the traditional byte boundaries. This type of 'bit-shifting' may allow for a higher granularity in the tool suites detection capability.

5.2.2 Diverse Applications

There are several potential applications and further enhancements of this research that I am interested in exploring. Some of these include computer forensics, code cloning and application authorship detection. This tool suite could easily be used by a computer forensics examiner to look at an entire hard drive that contains applications of unknown capabilities or an individual application. Code cloning is another interesting application of this research that is outside of the realm of computer security. Code cloning is a software engineering application that involves determining if a given application exists in multiple locations. Knowing this information is very helpful for software development firms as they try to update their software repositories. This tool again could easily be used to troll through a corpus of applications to determine if there exist similar applications to the one in question. These are just a few of the pathways that this method could be used for in the future.

5.3 Publication Plan

The following publications related to this research have been published:

- Travis Atkison, "Applying Randomized Projection to aid Prediction Algorithms in Detecting High-Dimensional Rogue Applications," *Proceedings of the 47th ACM Southeast Conference*, Clemson, SC, March 2009.
- Travis Atkison, "Using an Information Retrieval Technique to Discover Malicious Software," *Proceedings of the 5th Symposium on Risk Management and Cyber-Informatics*, Orlando, Florida, June 2008, International Institute of Informatics and Systemics, pp. 284 - 289.

The following publications related to this research are currently in review:

- Travis Atkison, "Randomized Projection versus Mutual Information: A Comparison Study on Rogue Application Detection," International Journal of Information Technology & Decision Making.
- Travis Atkison, "Using a Static Analysis Method in Detecting High-Dimensional Rogue Applications," the 16th International Static Analysis Symposium to be held in Los Angeles, CA, August 2009.

The following publications related to this research are planned:

- "Using Application Section Extraction and Randomized Projection to Improve Rogue Application Detection"
- "Randomized Projection Versus Principle Component Analysis: A Comparison Study on Rogue Application Detection"
- "Comparing Randomized Projection and Mutual Information Using Application Section Extraction to Improve Rogue Application Detection"
- "Comparing Randomized Matrix Creation Methods in a Rogue Application Detection Environment"

5.4 Candidate Journals

The following list of journals represents candidate outlets for future publication of this research.

1. ACM Transactions on Information and System Security - This journal is interested in security technologies including virus and Trojan horse detection which is the main thrust of this effort.
2. Computers and Security - This is one of the leading sources for applied research in computer security which is a direct fit with this research.
3. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - This research effort is an excellent example of direct application of data mining techniques and methodologies.
4. Journal of Machine Learning - The scope of this journal includes accounts of applications as well as experimental studies into the use of machine learning and data mining.
5. Journal of Information Assurance and Security - This research effort fits nicely within this journal regarding topics of detection of viruses and Trojan horses.
6. Journal of Information Assurance, Security and Protection - This journal's main focus is on information-based security. Our efforts fall directly in that realm.

7. IEEE Transactions on Information Forensics and Security - A direct application of this work lies in the forensics world which would be a good fit for this journal.
8. International Journal on Information Security - This journal has interest in direct applications of information security methods including virus and Trojan horse detection.
9. Information Retrieval - This journal is interested in application of information retrieval techniques to real-world problems such as our effort in rogue application detection.
10. Journal of Computer Security - This journal has interest in research related to protection against unauthorized disclosure or modification of information which fits nicely into the areas of this research effort.
11. IEEE Security and Privacy - This journal has a broad interest in computer security, including methods for detecting and securing infrastructure and the enterprise.

5.5 Candidate Conferences

The following list of conferences represents candidate outlets for future publication of this research.

1. IEEE Symposium on Security and Privacy - This conference interests include rogue application detection involving detection of viruses and Trojan horses.
2. USENIX Security Symposium - This conference has interest in many topics of security including the areas contained within this research effort.
3. SIAM International Conference on Data Mining - This conference has interest in applications of data mining techniques and methods.
4. ACM Symposium on Information, Computer and Communications Security - This effort has direct application to this conference in the area of rogue application detection.
5. Computer Security Applications Conference - As the name suggests, this conference has interest in direct application of computer security techniques and methods.
6. ACM SIGIR Information Retrieval Conference - This conference has interest in application of information retrieval techniques to real-world problems such as our effort in rogue application detection.
7. Computer Security Conference - This conference has interest in managing security risk which this research has direct applications to.

REFERENCES

[1] "Heaventools PE Explorer," http://www.heaventools.net (current 14 March 2009).

[2] "Malware," http://en.wikipedia.org/wiki/Malware (current 19 December 2007).

[3] T. Abou-Assaleh, N. Cercone, V. Keselj, and R. Sweidan, "Detection of New Malicious Code Using N-grams Signatures," *Proceedings of the 2nd Annual Conference on Privacy, Security and Trust*, New Brunswick, Canada, 2004, pp. 193–196.

[4] T. Abou-Assaleh, N. Cercone, V. Keselj, and R. Sweidan, "N-gram-based Detection of New Malicious Code," *Proceedings of the 28th Annual International Computer Software and Applications Conference, COMPSAC*, 2004, vol. 2.

[5] L. M. Adleman, "An Abstract Theory of Computer Viruses," *Proceedings of the 8th Annual International Cryptology Conference on Advances in Cryptology*, 1988, pp. 354–374.

[6] R. I. Arriaga and S. Vempala, "An algorithmic theory of learning: Robust concepts and random projection," *Machine Learning*, vol. 63, no. 2, 2006, pp. 161–182.

[7] T. Atkison, *Dimensionality Reduction Using a Randomized Projection Algorithm: Preliminary Results*, Tech. Rep. TR-CS-01-11, University of Maryland Baltimore County, 2001.

[8] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*, Addison Wesley, Harlow, England, 1999.

[9] U. Bayer, A. Moser, C. Kruegel, and E. Kirda, "Dynamic Analysis of Malicious Code," *Journal in Computer Virology*, vol. 2, 2006, pp. 67 – 77.

[10] R. Bellman, *Adaptive Control Processes: A Guided Tour*, Princeton University Press, 1961.

[11] J. Bergeron, M. Debbabi, J. Desharnais, M. M. Erhioui, Y. Lavoie, N. Tawbi, and M. Erhioui, "Static Detection of Malicious Code in Executable Programs," *Symposium on Requirements Engineering for Information Security*, Indianapolis, IN, 2001.

[12] E. Bingham and H. Mannila, "Random projection in dimensionality reduction: applications to image and text data," *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2001, pp. 245–250.

[13] R. A. Bosch and J. A. Smith, "Separating Hyperplanes and the Authorship of the Disputed Federalist Papers," *The American Mathematical Monthly*, vol. 105, no. 7, 1998, pp. 601–608.

[14] X. Chen, B. Francia, M. Li, B. McKinnon, and A. Seker, "Shared information and program plagiarism detection," *IEEE Transactions on Information Theory*, vol. 50, no. 7, 2004, pp. 1545–1551.

[15] M. Christodorescu and S. Jha, "Static analysis of executables to detect malicious patterns," *Proceedings of the 12th Conference on USENIX Security Symposium-Volume 12*, 2003, p. 12.

[16] M. Christodorescu, S. Jha, S. A. Seshia, D. Song, and R. E. Bryant, "Semantics-Aware Malware Detection," *2005 IEEE Symposium on Security and Privacy*, 2005, pp. 32–46.

[17] W. W. Cohen, "Learning Trees and Rules with Set-Valued Features," *AAAI/IAAI*, vol. 1, 1996, pp. 709–716.

[18] W. W. Cohen, A. Prieditis, and S. Russell, "Fast Effective Rule Induction," *Proceedings of the 12th International Conference on Machine Learning*, 1995, pp. 115–123.

[19] L. J. Cowen and C. E. Priebe, "Randomized non-linear projections uncover high-dimensional structure," *Advances in Applied Math*, vol. 19, 1997, pp. 319–331.

[20] S. Dasgupta and A. Gupta, *An elementary proof of the Johnson-Lindenstrauss Lemma*, Tech. Rep. TR-99-006, International Computer Science Institute, Berkley, CA, 1999.

[21] O. de Vel, A. Anderson, M. Corney, and G. Mohay, "Mining e-mail content for author identification forensics," *ACM SIGMOD Record*, vol. 30, no. 4, 2001, pp. 55–64.

[22] G. Florez-Larrahondo, Z. Liu, Y. S. Dandass, S. Bridges, and R. Vaughn, "Integrating Intelligent Anomaly Detection Agents into Distributed Monitoring Systems," *Journal of Information Assurance and Security*, vol. 1, 2006, pp. 59–77.

[23] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff, "A Sense of Self for Unix Processes," *Proceedings of the 1996 IEEE Symposium on Research in Security and Privacy*, 1996, pp. 120–128.

[24] G. Fung, "The disputed federalist papers: SVM feature selection via concave minimization," *Proceedings of the 2003 Conference on Diversity in Computing*, 2003, pp. 42–46.

[25] A. R. Gray, P. J. Sallis, S. G. MacDonell, and S. University of Otago Dept. of Information, *Software Forensics: Extending Authorship Analysis Techniques to Computer Programs*, Dept. of Information Science, University of Otago, 1997.

[26] O. Henchiri and N. Japkowicz, "A Feature Selection and Evaluation Scheme for Computer Virus Detection," *6th International Conference on Data Mining, ICDM'06*, 2006, pp. 891–895.

[27] D. I. Holmes, "The Analysis of Literary Style," *Journal of the Royal Statistical Society. Series A*, vol. 148, no. 4, 1985, pp. 328–341.

[28] G. Hristescu and M. Farach-Colton, *Cluster-preserving embedding of proteins*, Tech. Rep. TR 99-50, Rutgers University Center for Discrete Mathematics and Computer Science (DIMACS), 1999.

[29] P. Indyk and R. Motwani, "Approximate nearest neighbors: towards removing the curse of dimensionality," *Proceedings of the 13th Annual ACM Symposium on Theory of Computing*, 1998, pp. 604–613.

[30] W. B. Johnson and J. Lindenstrauss, "Extensions of Lipschitz mappings into a Hilbert space," *Contemporary Mathematics*, vol. 26, 1984, pp. 189–206.

[31] S. Kaski, "Dimensionality Reduction by Random Mapping: Fast Similarity Computation for Clustering," *The 1998 IEEE International Joint Conference on Neural Networks. IEEE World Congress on Computational Intelligence*, vol. 1, 1998.

[32] J. O. Kephart, G. B. Sorkin, W. C. Arnold, D. M. Chess, G. J. Tesauro, and S. R. White, "Biologically inspired defenses against computer viruses," *Proceedings of the 14th International Joint Conference on Artificial Intellgence*, San Francisco, CA, 1995, pp. 985–996.

[33] V. Keselj, F. Peng, N. Cercone, and C. Thomas, "N-gram-based Author Profiles for Authorship Attribution," *Proceedings of the Conference Pacific Association for Computational Linguistics*, Dalhousie University, Halifax, Nova Scotia, Canada, 2003, pp. 255–264.

[34] J. M. Kleinberg, "Two algorithms for nearest-neighbor search in high dimensions," *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, 1997, pp. 599–608.

[35] C. Ko, G. Fink, and K. Levitt, "Automated detection of vulnerabilities in privileged programs by execution monitoring," *Proceedings of the 10th Annual Computer Security Applications Conference*, 1994, pp. 134–144.

[36] J. Z. Kolter and M. A. Maloof, "Learning to Detect Malicious Executables in the Wild," *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Seattle, WA, 22-25 August 2004, pp. 470–478, ACM Press.

[37] J. Z. Kolter and M. A. Maloof, "Learning to Detect and Classify Malicious Executables in the Wild," *The Journal of Machine Learning Research*, vol. 7, 2006, pp. 2721–2744.

[38] M. Kurimo, "Indexing Audio Documents by using Latent Semantic Analysis and SOM," *Kohonen Maps*, 1999, pp. 363–374.

[39] J. Lin and D. Gunopulos, "Dimensionality reduction by random projection and latent semantic indexing," *Proceedings of the Text Mining Workshop, at the 3rd SIAM International Conference on Data Mining*, 2003.

[40] N. Linial, E. London, and Y. Rabinovich, "The geometry of graphs and some of its algorithmic applications," *Combinatorica*, vol. 15, no. 2, 1995, pp. 215–245.

[41] R. W. Lo, K. N. Levitt, and R. A. Olsson, "MCF: a malicious code filter," *Computers & Security*, vol. 14, no. 6, 1995, pp. 541–566.

[42] H. Mannila and J. K. Seppänen, "Finding similar situations in sequences of events," *First SIAM International Conference on Data Mining*, 2001.

[43] C. Marceau, "Characterizing the Behavior of a Program Using Multiple-Length N-grams," *Proceedings of the 2000 Workshop on New Security Paradigms*, Ballycotton, County Cork, Ireland, 2000, ACM.

[44] G. McGraw and G. Morrisett, "Attacking malicious code: a report to the Infosec Research Council," *IEEE Software*, vol. 17, no. 5, 2000, pp. 33–41.

[45] M. Moffie, W. Cheng, D. Kaeli, and Q. Zhao, "Hunting Trojan Horses," *Proceedings of the 1st Workshop on Architectural and System Support for Improving Software Dependability*, San Jose, CA, 21 October 2006, pp. 12–17, ACM Press.

[46] A. Moser, C. Kruegel, and E. Kirda, "Exploring Multiple Execution Paths for Malware Analysis," *IEEE Symposium on Security and Privacy, SP'07*, 2007, pp. 231–245.

[47] C. H. Papadimitriou, P. Raghavan, H. Tamaki, and S. Vempala, "Latent Semantic Indexing: A Probabilistic Analysis," *Journal of Computer and System Sciences*, vol. 61, no. 2, 2000, pp. 217–235.

[48] M. D. Preda, M. Christodorescu, S. Jha, and S. Debray, "A Semantics-Based Approach to Malware Detection," *Proceedings of the 34th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Nice, France, 17-19 January 2007, pp. 377–388, ACM Press.

[49] S. Rawat, V. P. Gulati, A. K. Pujari, and V. R. Vemuri, "Intrusion Detection using Text Processing Techniques with a Binary-Weighted Cosine Metric," *Journal of Information Assurance and Security*, vol. 1, no. 1, 2006, pp. 43–50.

[50] D. K. S. Reddy and A. K. Pujari, "N-gram analysis for computer virus detection," *Journal in Computer Virology*, vol. 2, no. 3, 2006, pp. 231–239.

[51] R. Richardson, *2007 CSI Computer Crime and Security Survey*, Tech. Rep., Computer Security Institute, 2007.

[52] H. Rogers, *Theory of Recursive Functions and Effective Computability*, McGraw-Hill, 1967.

[53] M. Sabordo, S. Y. Chai, M. J. Berryman, and D. Abbott, ""Who wrote the 'Letter to the Hebrews'?": data mining for detection of text authorship," *Proceedings of SPIE*, 2005, vol. 5649, p. 513.

[54] P. Sallis, A. Aakjaer, and S. MacDonell, "Software Forensics: Old Methods for a New Science," *Proceedings of SE: E&P'96 (Software Engineering: Education and Practice)*, 1996, pp. 367–371.

[55] M. Salois and R. Charpentier, "Dynamic Detection of Malicious Code in COTS Software," *Commercial Off-the-Shelf Products in Defense Applications "The Ruthless Pursuit of COTS"*, vol. 1, 2000.

[56] G. Salton and C. Buckley, "Term-weighting approaches in automatic text retrieval," *Information Processing and Management: an International Journal*, vol. 24, no. 5, 1988, pp. 513–523.

[57] G. Salton, A. Wong, and C. S. Yang, "A vector space model for automatic indexing," *Communications of the ACM*, vol. 18, no. 11, 1975, pp. 613–620.

[58] T. Sandeep and M. P. Jignesh, "Estimating the Selectivity of tf-idf based Cosine Similarity Predicates," *ACM SIGMOD Record*, vol. 36, no. 2, 2007, pp. 7–12.

[59] M. Schultz, E. Eskin, E. Zadok, and S. Stolfo, "Data Mining Methods for Detection of New Malicious Executables," *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, 14-16 May 2001, pp. 38–49, IEEE.

[60] G. Shafer, "A Mathematical Theory of Evidence," *Princeton, NJ*, 1976.

[61] A. Singhal, "Modern Information Retrieval: A Brief Overview," *Bulletin of the Technical Committee on Data Engineering*, vol. 24, no. 4, 2001, pp. 35–43.

[62] S. M. Sladaritz Sr, "About Heuristics," SANS Information Security Reading Room, March 2002.

[63] P. Soucy and G. Mineau, "Beyond TFIDF Weighting for Text Categorization in the Vector Space Model," *19th International Joint Conference on Artificial Intelligence*, Edinburgh, Scotland, 30 July - 5 August 2005, pp. 1130–1135.

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

[65] Symantec, *Symantec Internet Security Threat Report: Trends for January 05 - June 05*, Tech. Rep., Symantec, September 2005.

[66] G. J. Tesauro, J. O. Kephart, and G. B. Sorkin, "Neural networks for computer virus recognition," *IEEE Intelligent Systems and Their Applications*, vol. 11, no. 4, 1996, pp. 5–6.

[67] L. Todorovski and S. Dzeroski, "Combining multiple models with meta decision trees," *Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery*, 2000, p. 5464.

[68] A. Vasudevan and R. Yerraballi, "Cobra: Fine-grained Malware Analysis using Stealth Localized-Executions," *IEEE Symposium on Security and Privacy*, 2006.

[69] S. S. Vempala, *The Random Projection Method*, American Mathematical Society, 2004.

[70] K. L. Verco and M. J. Wise, "Software for detecting suspected plagiarism: Comparing structure and attribute-counting systems," *Proceedings of 1st Australian Conference on Computer Science Education*, Sydney, July 1996.

[71] J. von Neumann, "Theory of Self-Reproducing Automata," *Part 1: Transcripts of lectures given at the University of Illinois, December 1949*, A. W. Burks, ed., 1966.

[72] C. Willems, T. Holz, and F. Freiling, "Toward Automated Dynamic Malware Analysis Using CWSandbox," *IEEE Security & Privacy Magazine*, vol. 5, no. 2, 2007, pp. 32–39.

[73] Y. Yang and J. O. Pedersen, "A Comparative Study on Feature Selection in Text Categorization," *Proceedings of the 14th International Conference on Machine Learning*, 1997, pp. 412–420.

[74] B. Zhang, J. Yin, J. Hao, S. Wang, D. Zhang, and W. Tang, "New Malicious Code Detection Based on N-gram Analysis and Rough Set Theory," *International Conference on Computational Intelligence and Security*, 2006, vol. 2.

APPENDIX A

ADLEMAN'S THEOREM FOR DETECTING VIRUSES

For all Gödel numberings of the partial recursive functions $\{\phi_i\}$

$$V = \{i|\phi_i \text{ is a virus}\} \text{ is } \Pi_2 - complete$$

Proof

Let $T = \{i|\phi_i \text{ is a total}\}$. It is well known (§13 and §14 in [52]) that T is $\Pi_2 - complete$.

To establish that $T \leq_1 V$, let $j \in V$ (for example let j be an index for the identity function) and consider the function $g : N \to N$ such that for $i, y \in N$:

$$g(i,y) = \begin{cases} \phi_j(y) & if\ \phi_i(y)\ \downarrow \\ \uparrow & otherwise \end{cases}$$

Then g is a partial recursive function. Let k be an index for g, and let $f : N \to N$, be such that:

$$(\forall i \in N)[f(i) = s(k,1,i)]$$

where s is as in the $s - m - n$ theorem [52].

Then f is a total recursive function and:

$$(\forall i, y \in N)\left[\phi_{f(i)}(y) = \phi_{s(k,1,i)}(y) = \phi_k(i,y) = g(i,y) = \begin{cases} \phi_j(y) & if\ \phi_i(y)\ \downarrow \\ \uparrow & otherwise \end{cases}\right]$$

It follows that:

$$i \in T \Leftrightarrow f(i) \in V$$

Thus $T \leq_m V$. It follows, as in §7.2 in [52], that $T \leq_1 V$ as desired.

To establish that $V \in \Pi_2$, consider the following formula for V which arises directly from the definition of virus:

$$
\begin{array}{ll}
(\forall j)(\exists k,t) & [H(i,j,k,t)] \\
\& & \\
(\forall <d,p>) & [(\forall j_1,k_1,t_1) \\
 & [H(i,j_1,k_1,t_1) \Rightarrow \\
 & (\forall <e,q>,t_2)[\neg H(k_1,<d,p>,<e,p>,t_2)]] \\
 & or \\
 & (\forall j_1,k_1,t_1,j_2,k_2,t_2) \\
 & [[H(i,j_1,k_1,t_1)\ \&\ H(i,j_2,k_2,t_2)] \Rightarrow \\
 & (\exists <e,q>,t_3,t_4) \\
 & [H(k_1,<d,p>,<e,q>,t_3)\ \& \\
 & H(k_2,<d,p>,<e,q>,t_4)]] \\
 & or \\
 & (\forall j_1,k_1,t_1,<e,q>,t_2) \\
 & [[H(i,j_1,k_1,t_1)\ \&\ H(j_1,<d,p>,<e,q>,t_2)] \Rightarrow \\
 & (\exists <e',q'>,t_3,t_4) \\
 & [H(k_1,<d,p>,<e',q'>,t_3)\ \& \\
 & L(i,<e,q>,<e',q'>,t_4)] \\
 & \& \\
 & H(i,j_1,k_1,t_1)\ \&\ H(k_1,<d,p>,<e,q>,t_2)] \Rightarrow \\
 & (\exists <e',q'>,t_3,t_4) \\
 & [H(j,<d,p>,<e',q'>,t_3)\ \& \\
 & L(i,<e',q'>,<e,q>,t_4)]]]
\end{array}
$$

Where H is a 'step counting' predicate for $\{\phi_i\}$ such that:

$$
\begin{array}{ll}
(\forall i,j,k) & \\
if\ \phi_i(j)=k\ then & (\exists t)[H(i,j,k,t)] \\
if\ \phi_i(j)\neq k\ then & (\forall t)[\neg H(i,j,k,t)]
\end{array}
$$

And where L is a predicate for $\{\phi_i\}$ such that:

$$
\begin{array}{ll}
(\forall i,<e,q>,<e',q'>,t) & \\
if <e,q>\sim^{\phi_i}<e',q'>\ then & (\exists t)[L(i,<e,q>,<e',q'>,t)] \\
if <e,q>\nsim^{\phi_i}<e',q'>\ then & (\forall t)[\neg L(i,<e,q>,<e',q'>,t)]
\end{array}
$$

Since for all acceptable Gödel numberings of the partial recursive functions $\{\phi_i\}$ it is easily seen that there exist recursive predicates H and L as above, it follows that $V \in \Pi_2$.

APPENDIX B

EXPERIMENTAL RESULTS GRAPHS

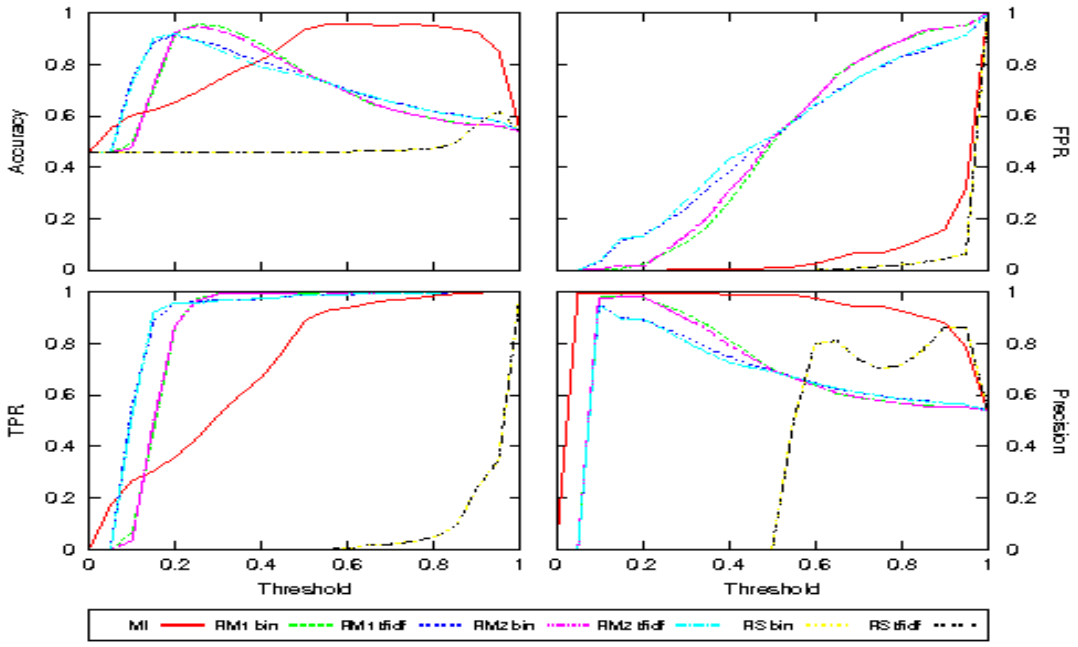

Figure B.1

3-gram, Whole portion data set, 500-features

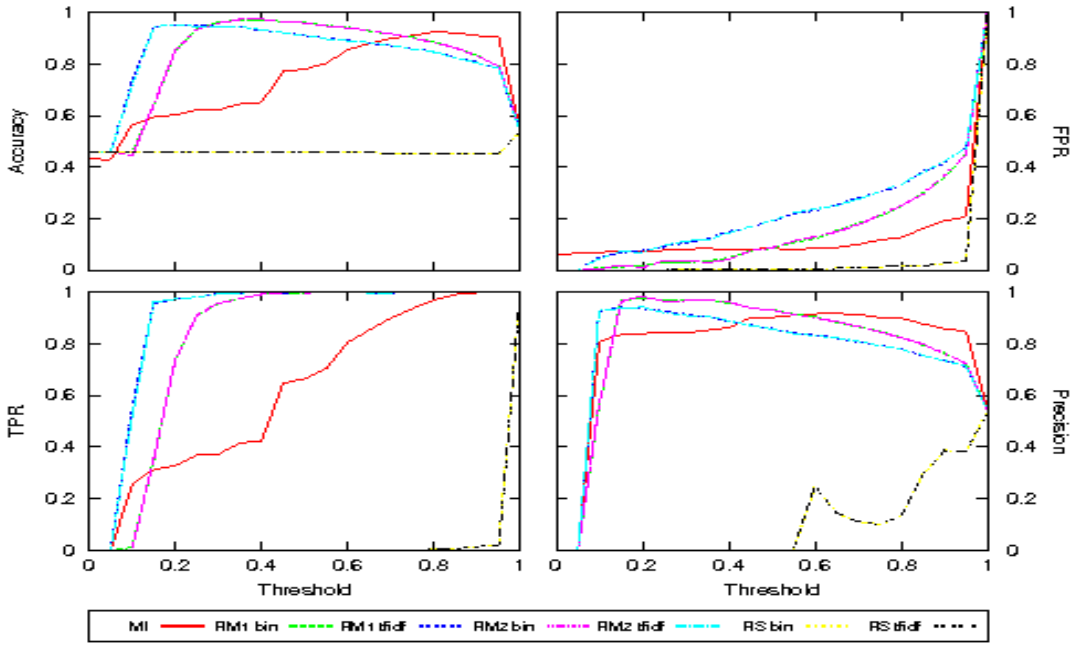

Figure B.2

3-gram, Data portion data set, 500-features

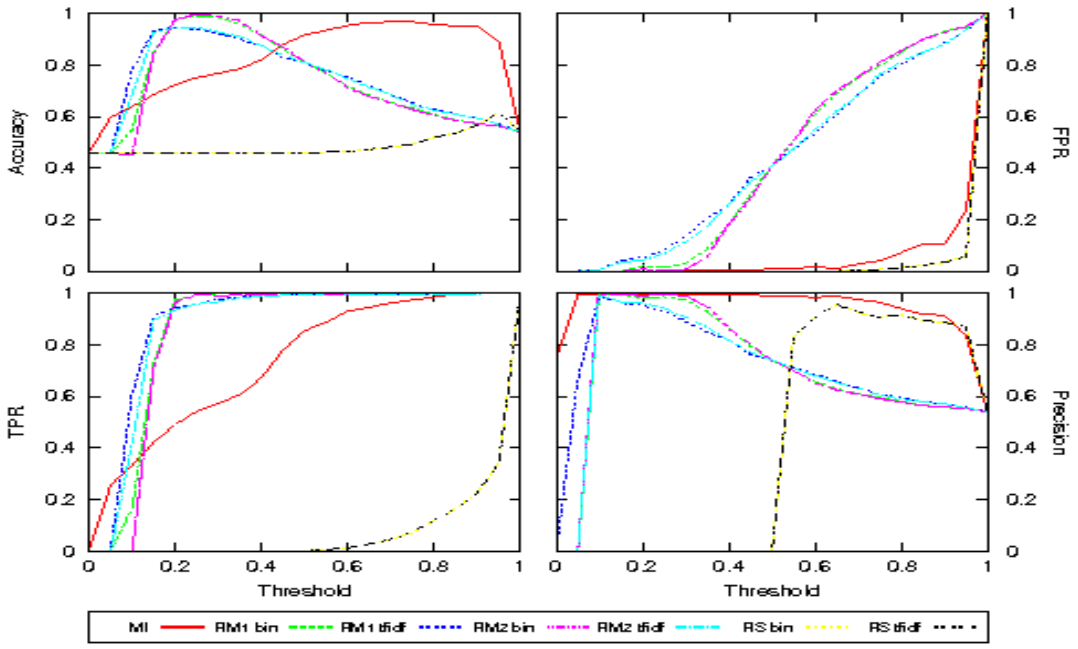

Figure B.3

3-gram, Code portion data set, 500-features

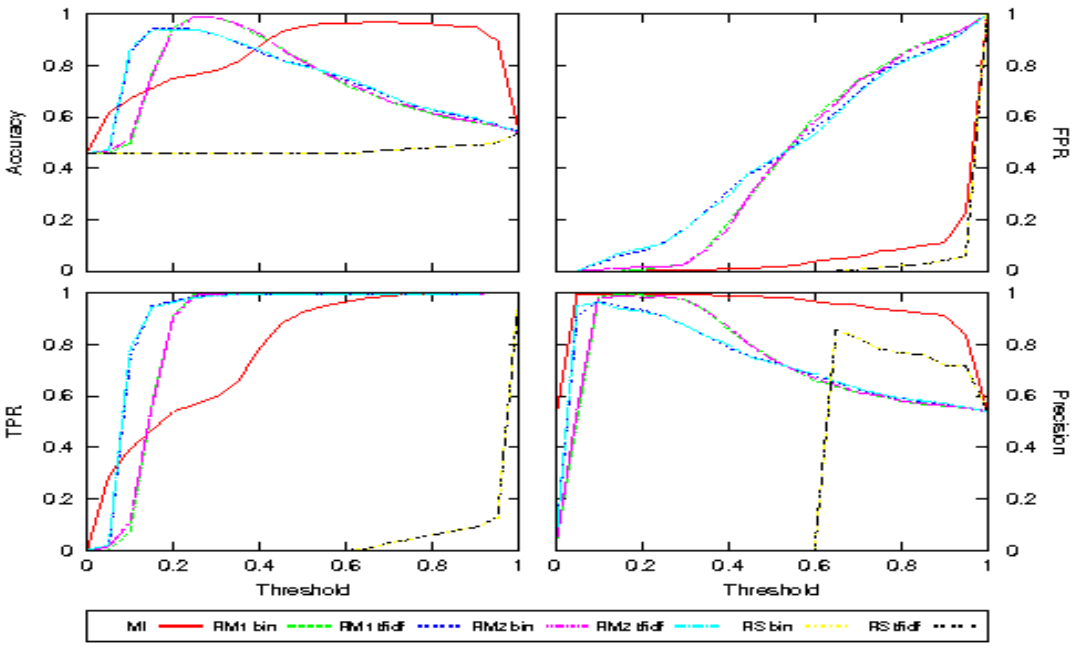

Figure B.4

3-gram, Combination portion data set, 500-features

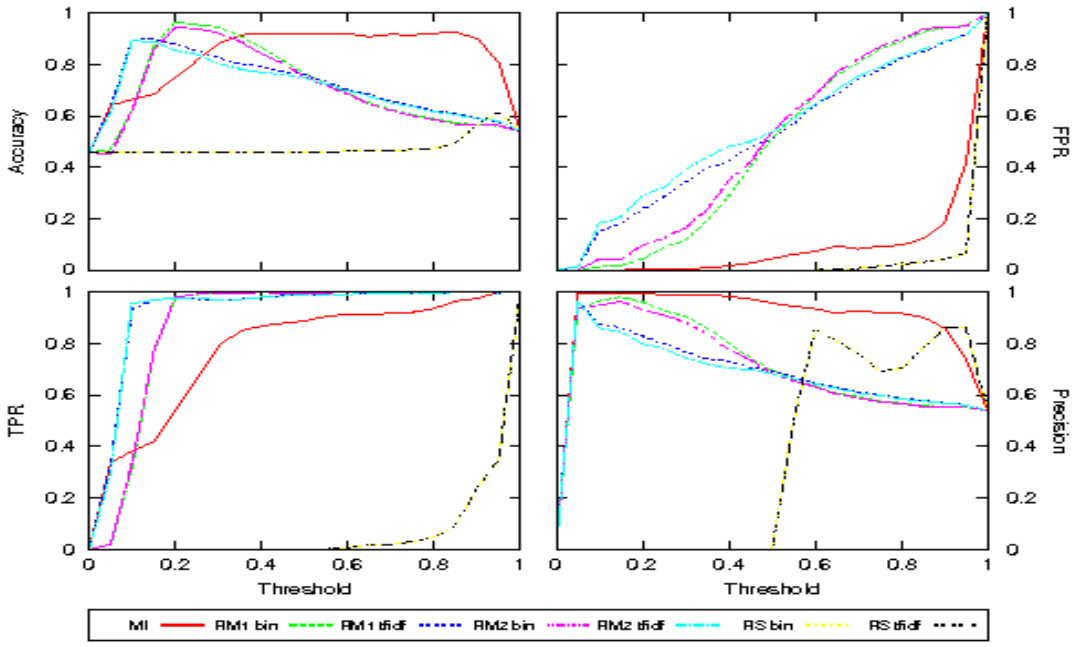

Figure B.5

3-gram, Whole portion data set, 1000-features

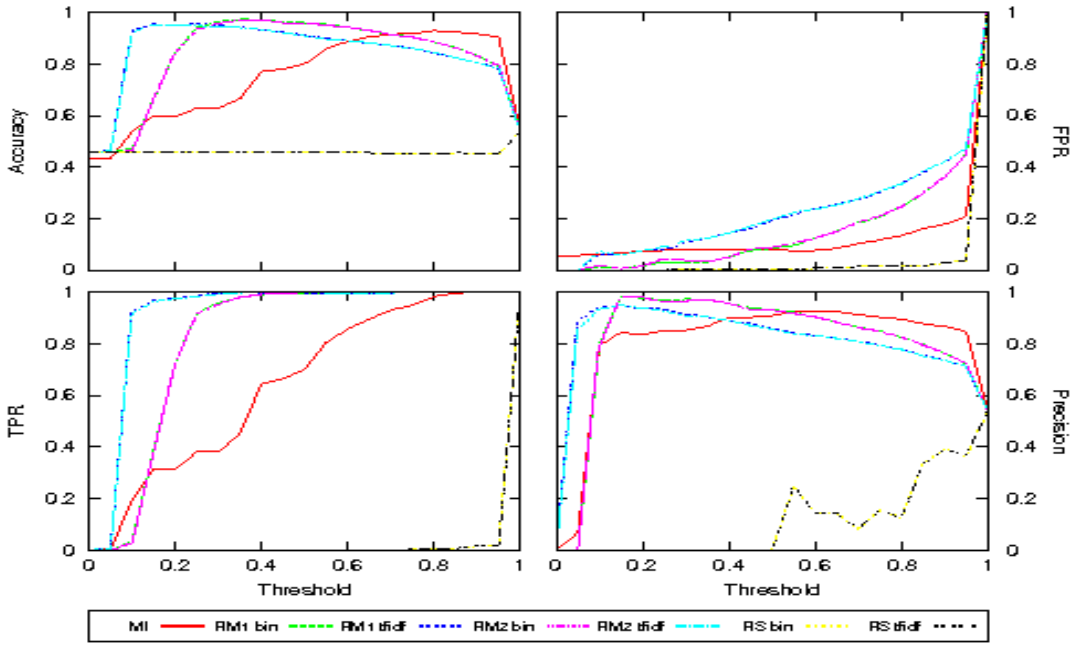

Figure B.6

3-gram, Data portion data set, 1000-features

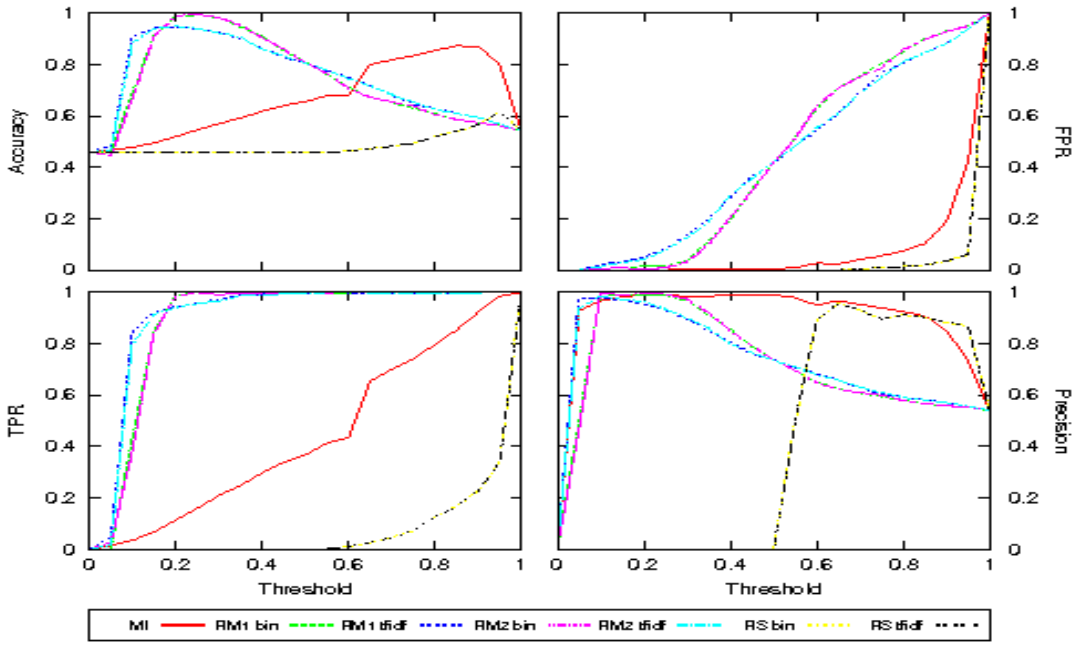

Figure B.7

3-gram, Code portion data set, 1000-features

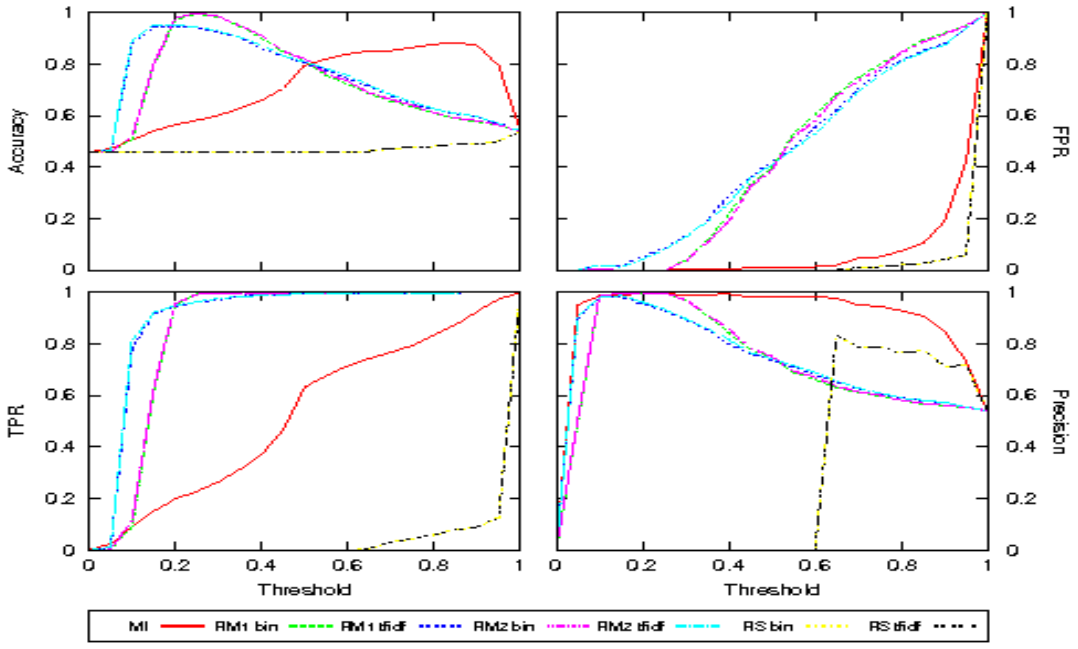

Figure B.8

3-gram, Combination portion data set, 1000-features

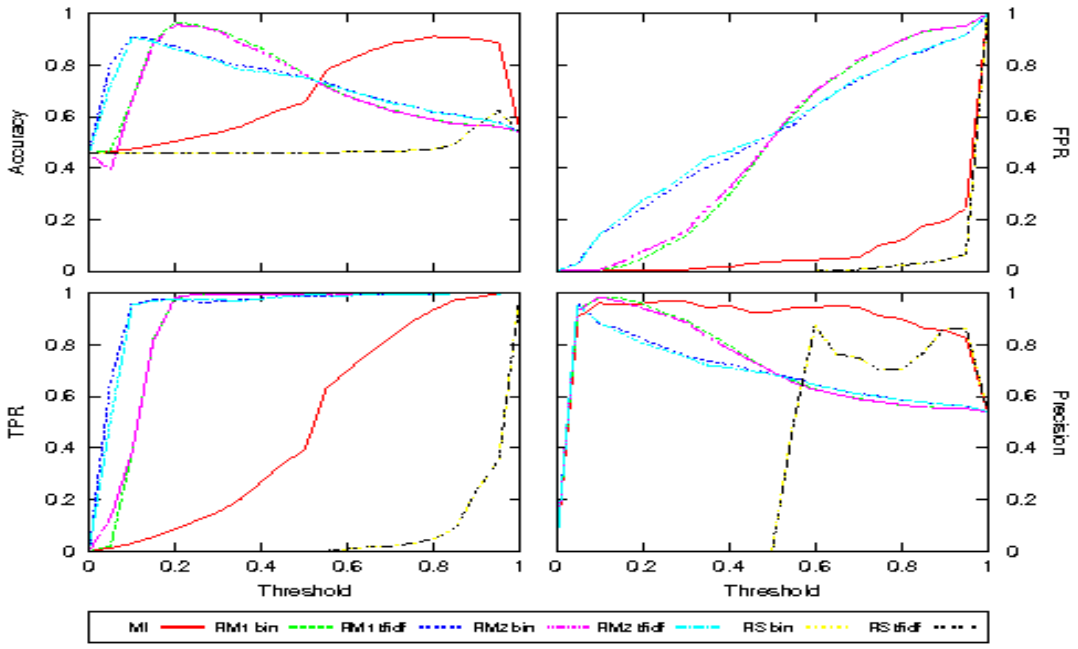

Figure B.9

3-gram, Whole portion data set, 1500-features

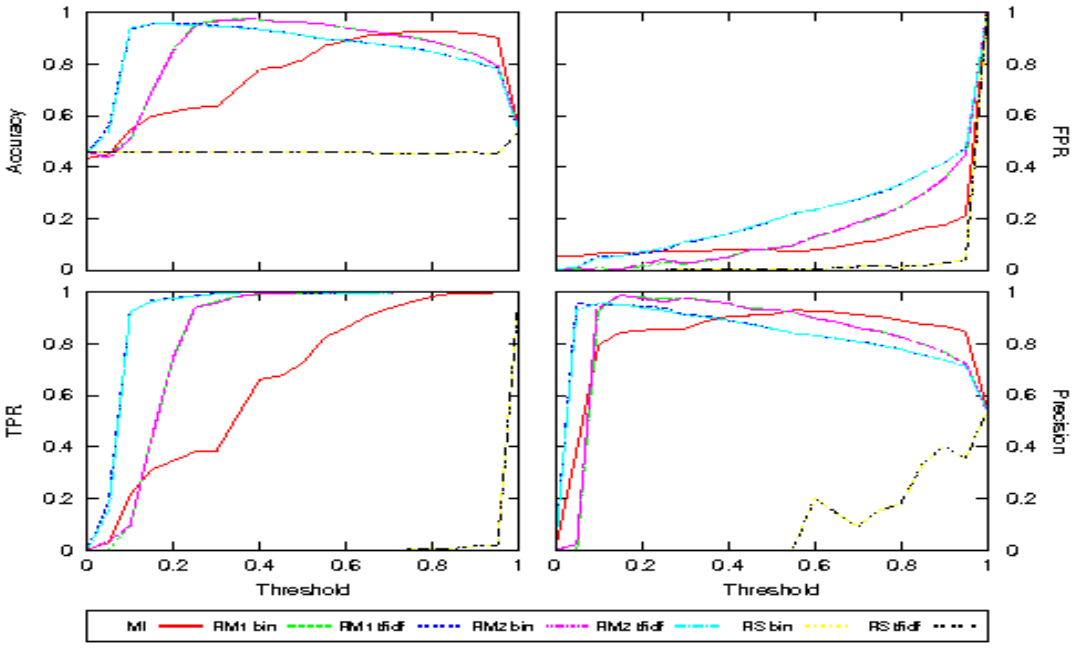

Figure B.10

3-gram, Data portion data set, 1500-features

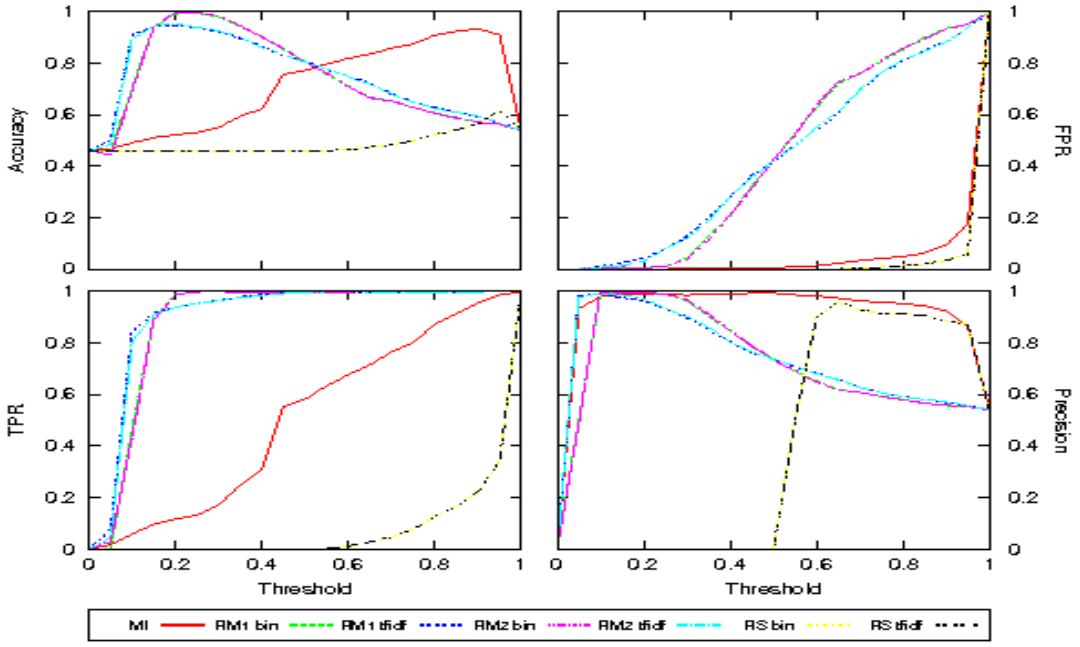

Figure B.11

3-gram, Code portion data set, 1500-features

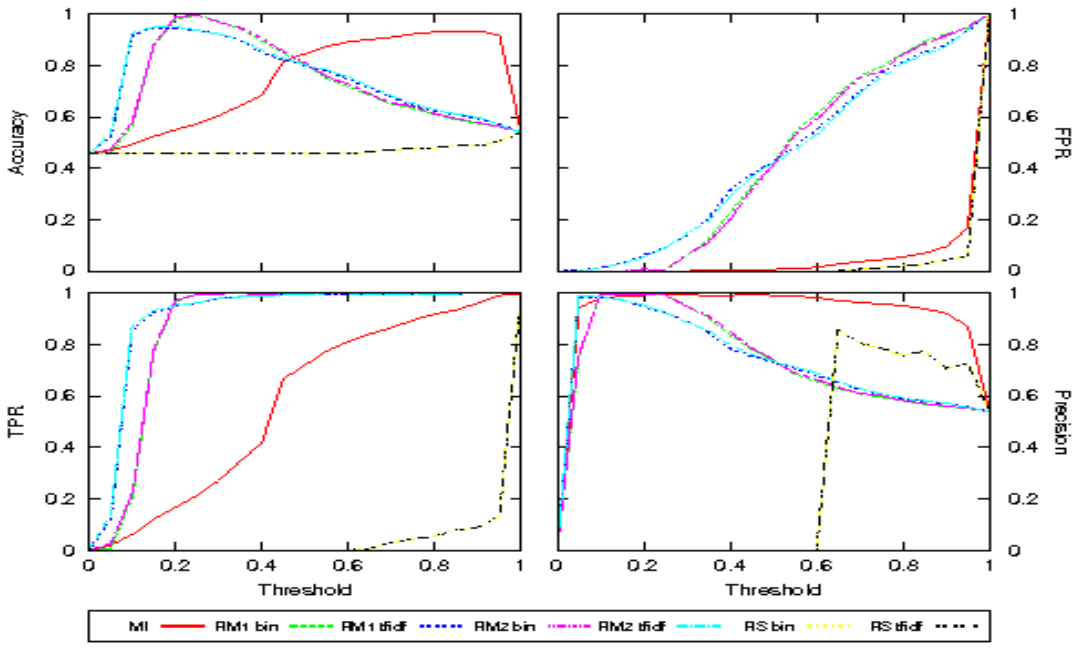

Figure B.12

3-gram, Combination portion data set, 1500-features

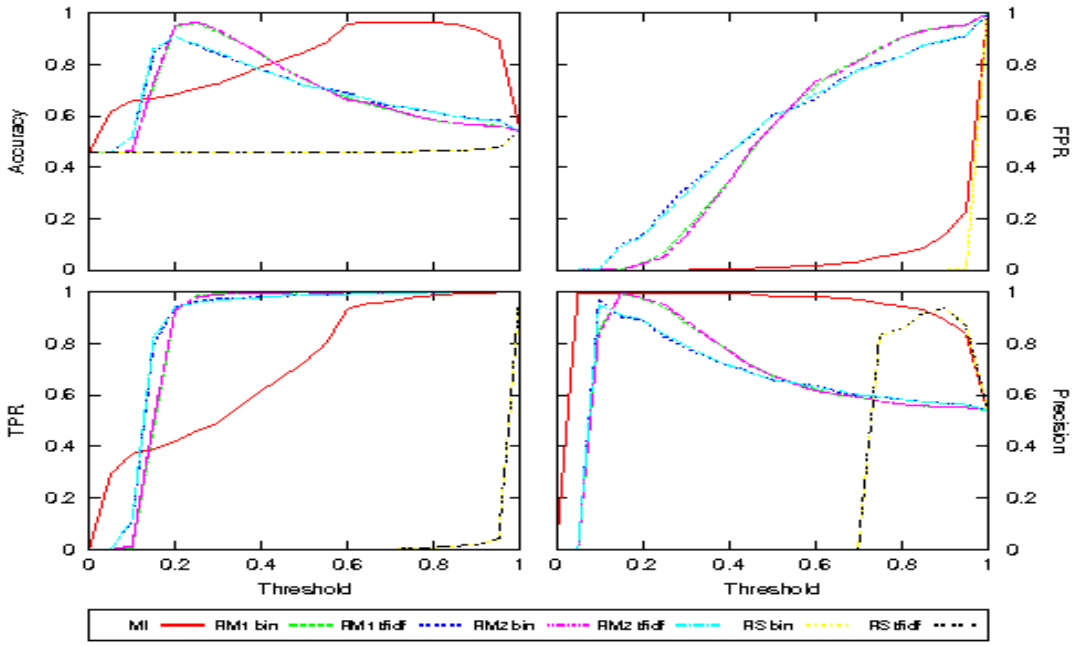

Figure B.13

4-gram, Whole portion data set, 500-features

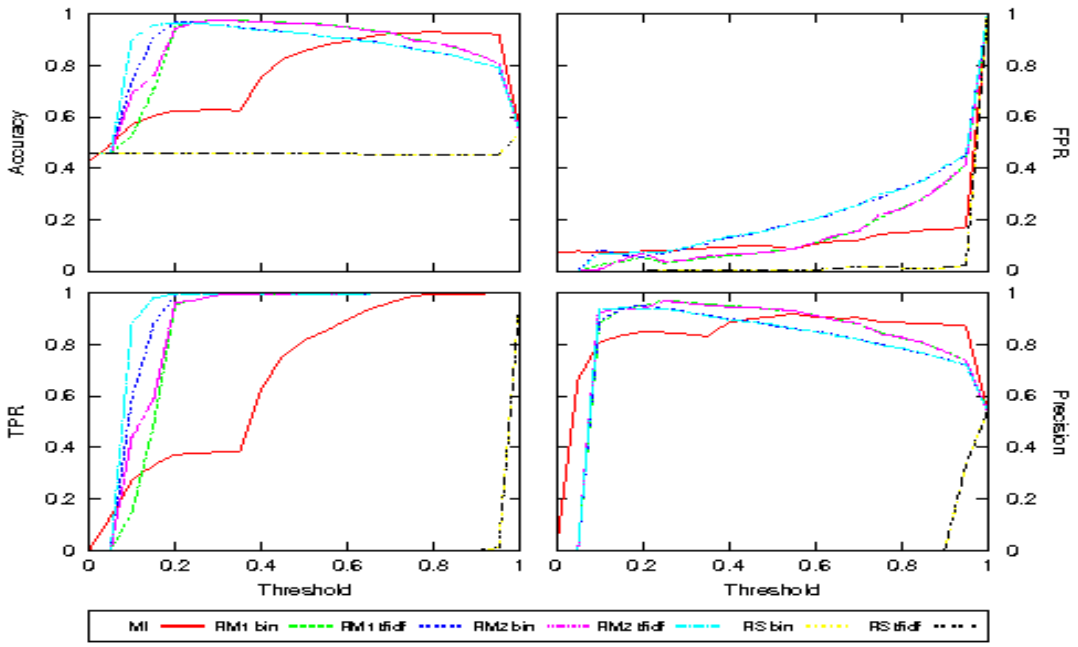

Figure B.14

4-gram, Data portion data set, 500-features

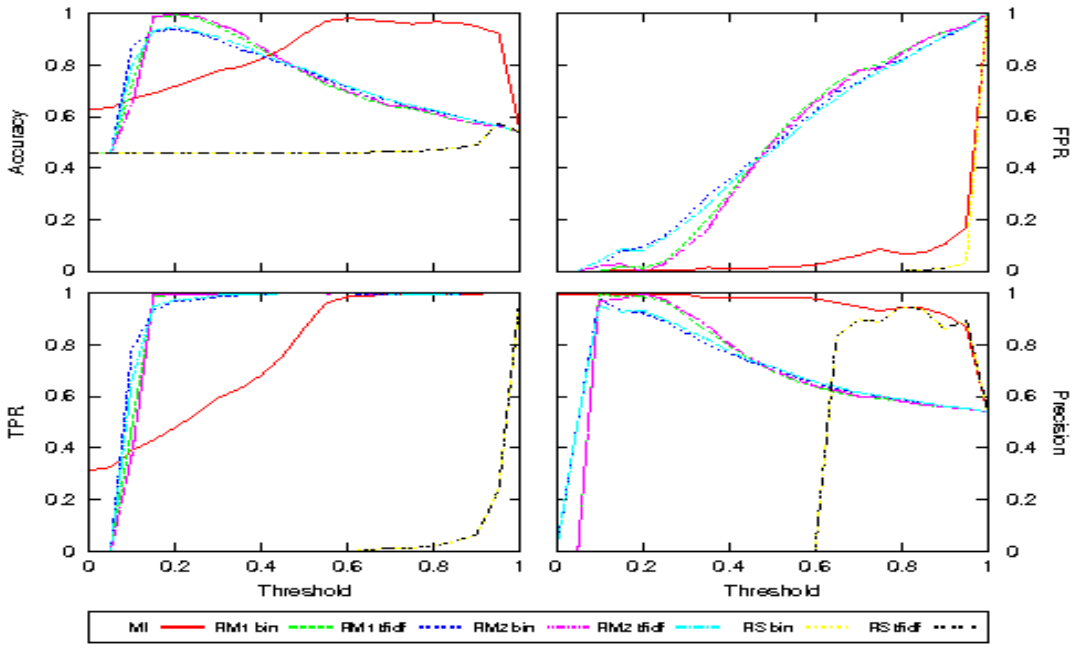

Figure B.15

4-gram, Code portion data set, 500-features

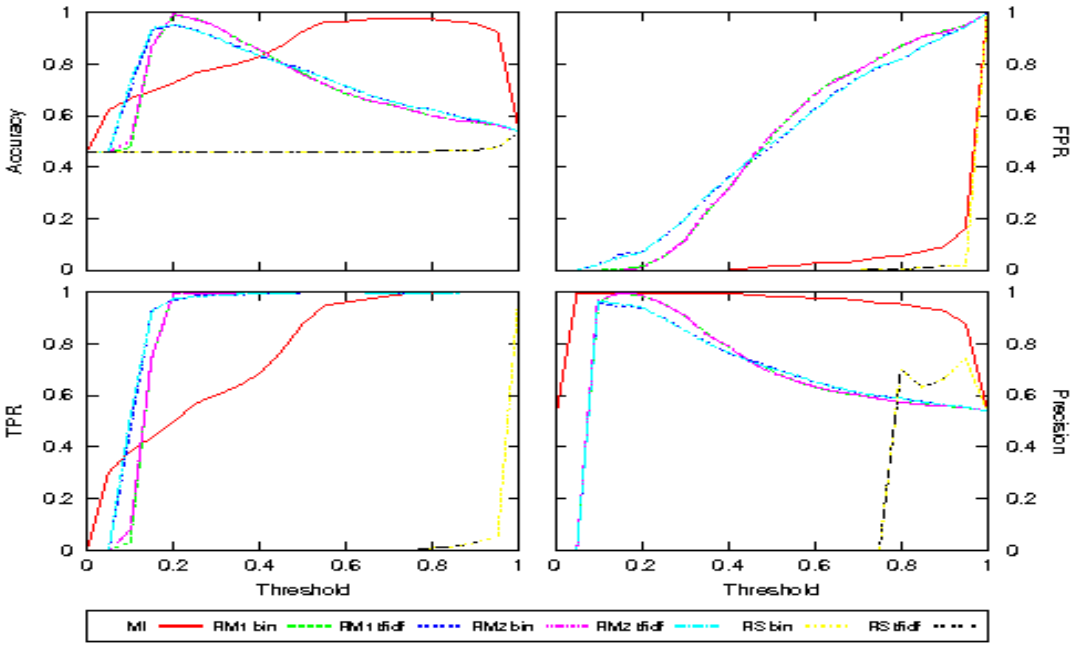

Figure B.16

4-gram, Combination portion data set, 500-features

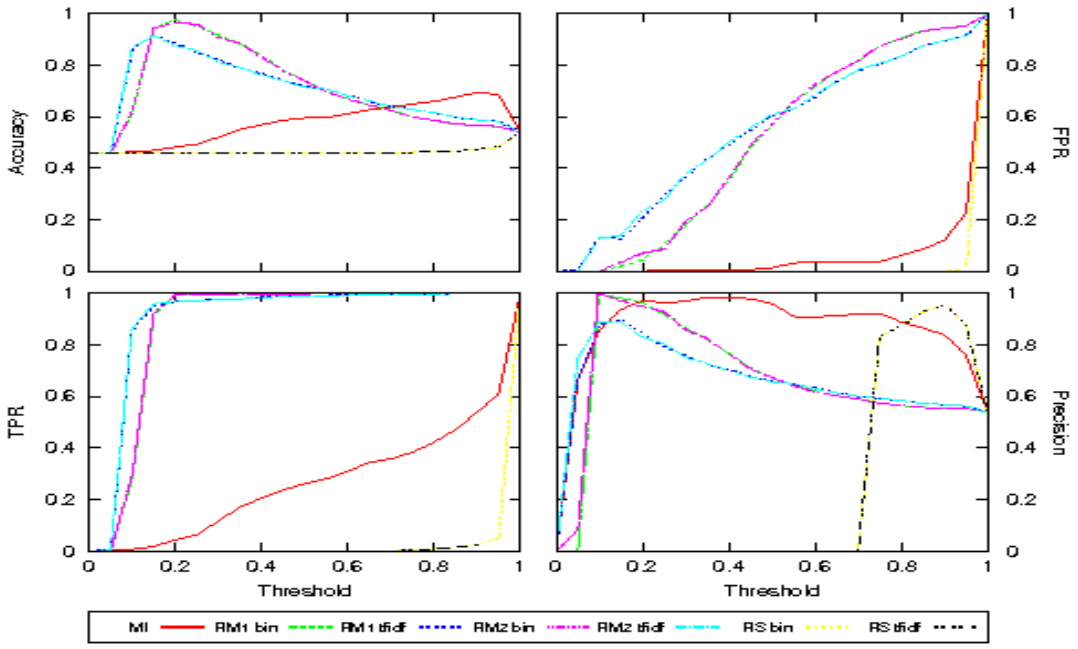

Figure B.17

4-gram, Whole portion data set, 1000-features

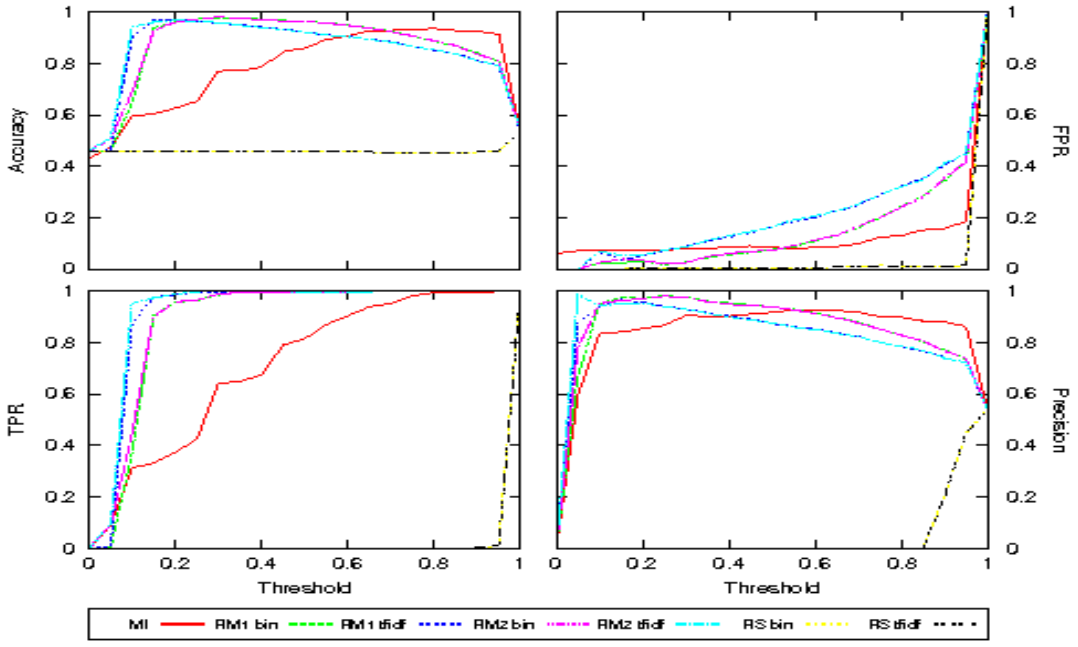

Figure B.18

4-gram, Data portion data set, 1000-features

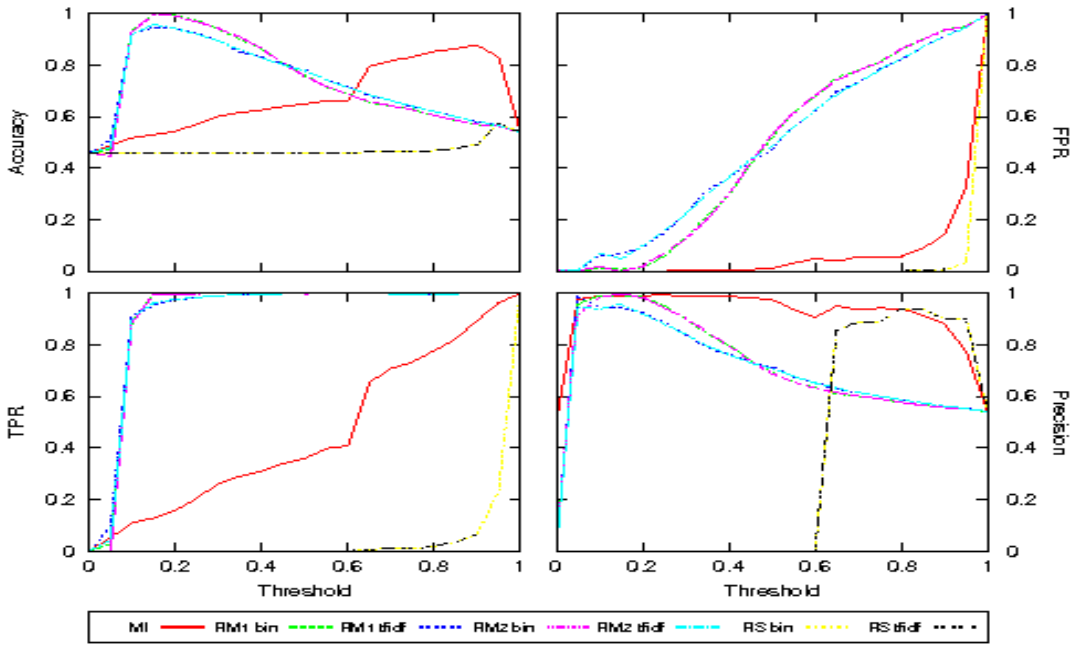

Figure B.19

4-gram, Code portion data set, 1000-features

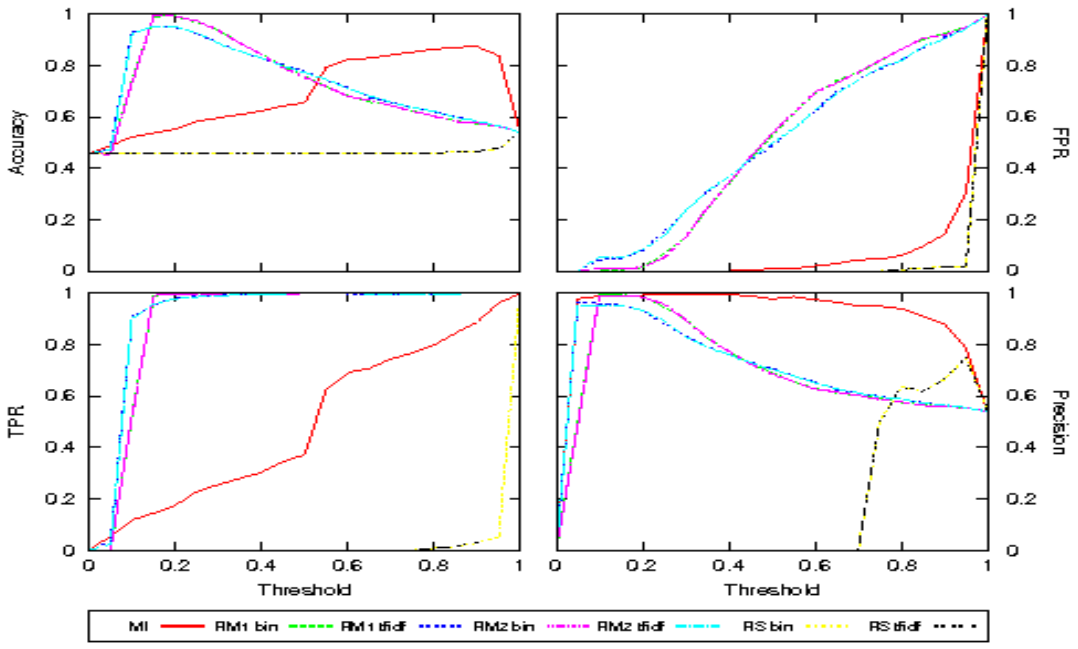

Figure B.20

4-gram, Combination portion data set, 1000-features

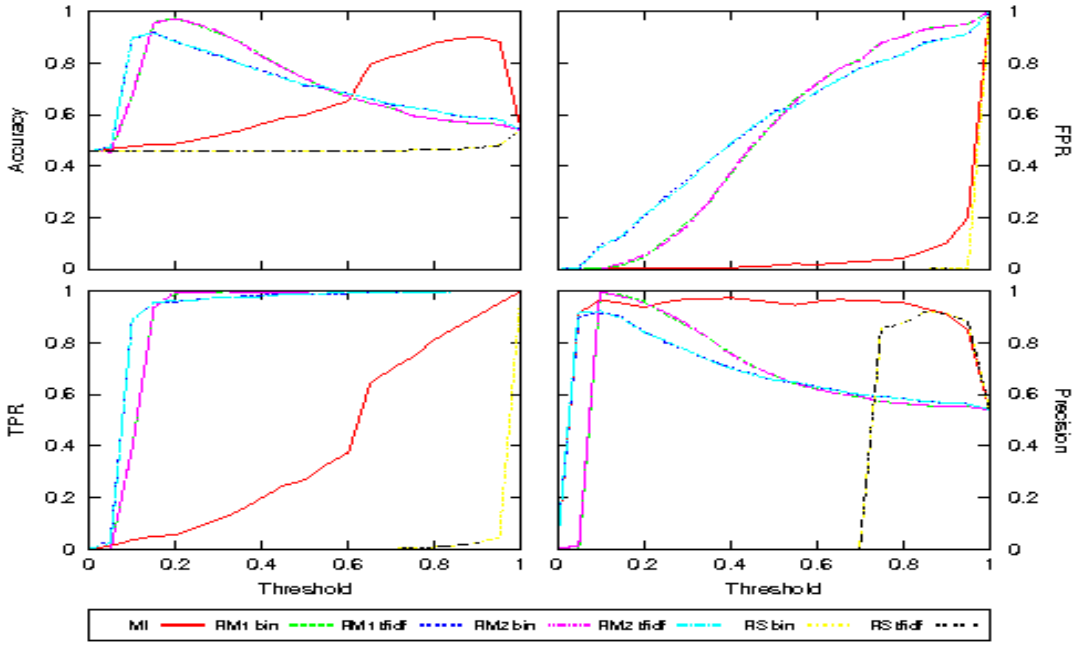

Figure B.21

4-gram, Whole portion data set, 1500-features

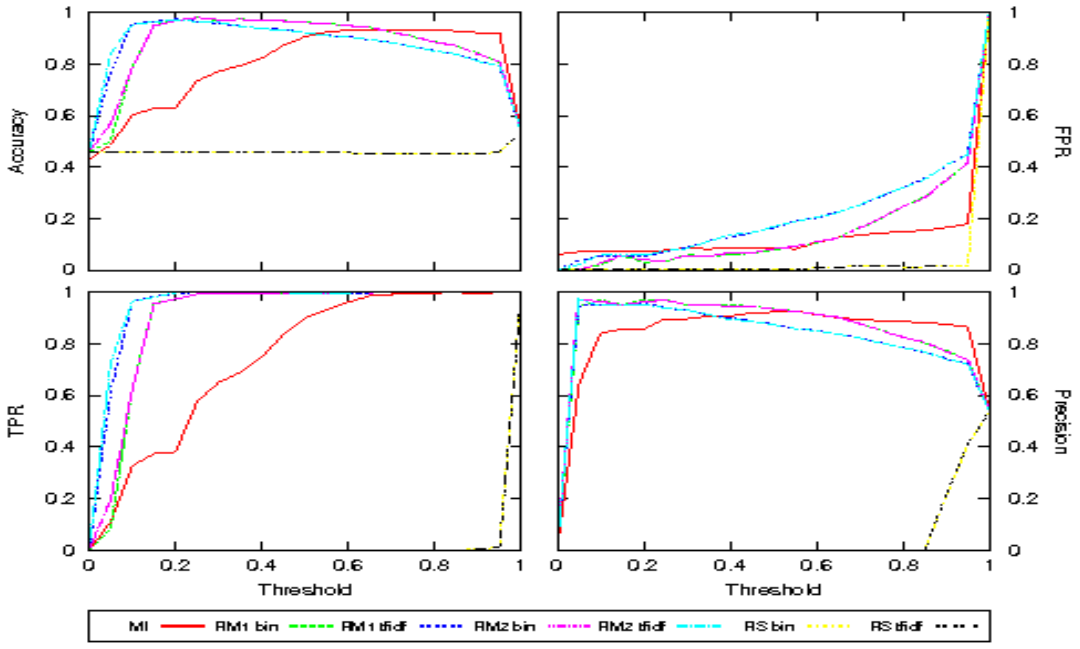

Figure B.22

4-gram, Data portion data set, 1500-features

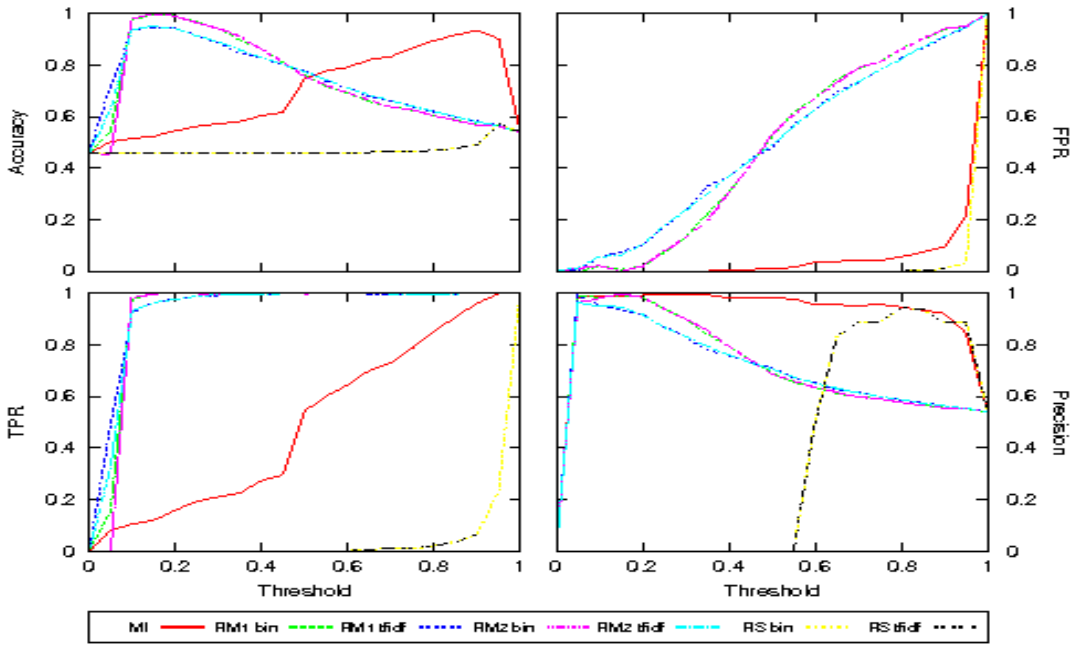

Figure B.23

4-gram, Code portion data set, 1500-features

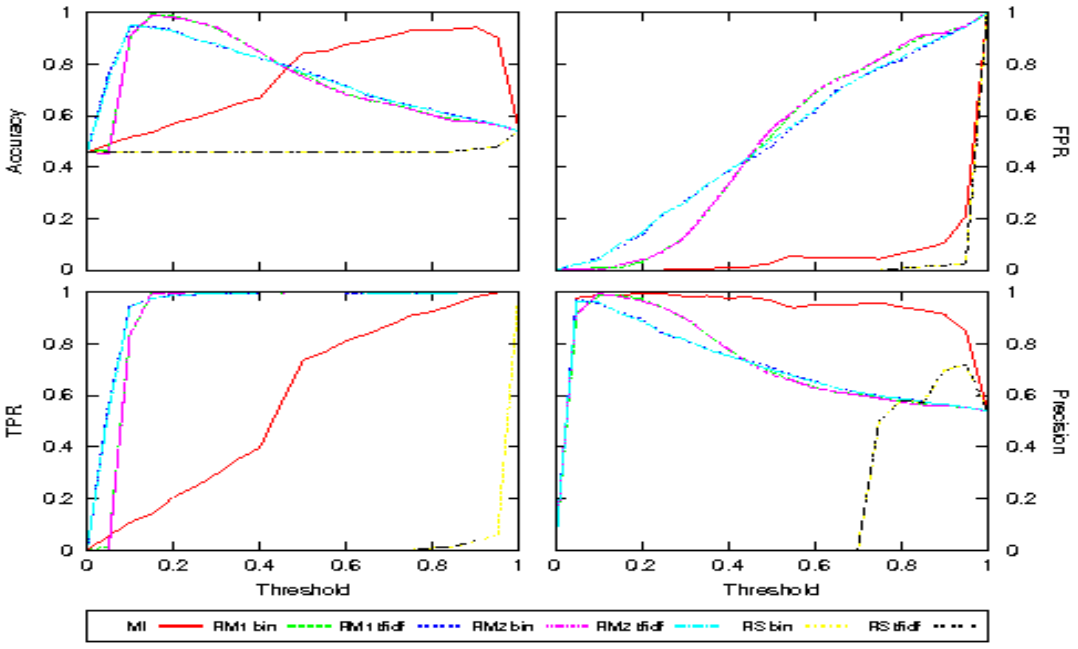

Figure B.24

4-gram, Combination portion data set, 1500-features

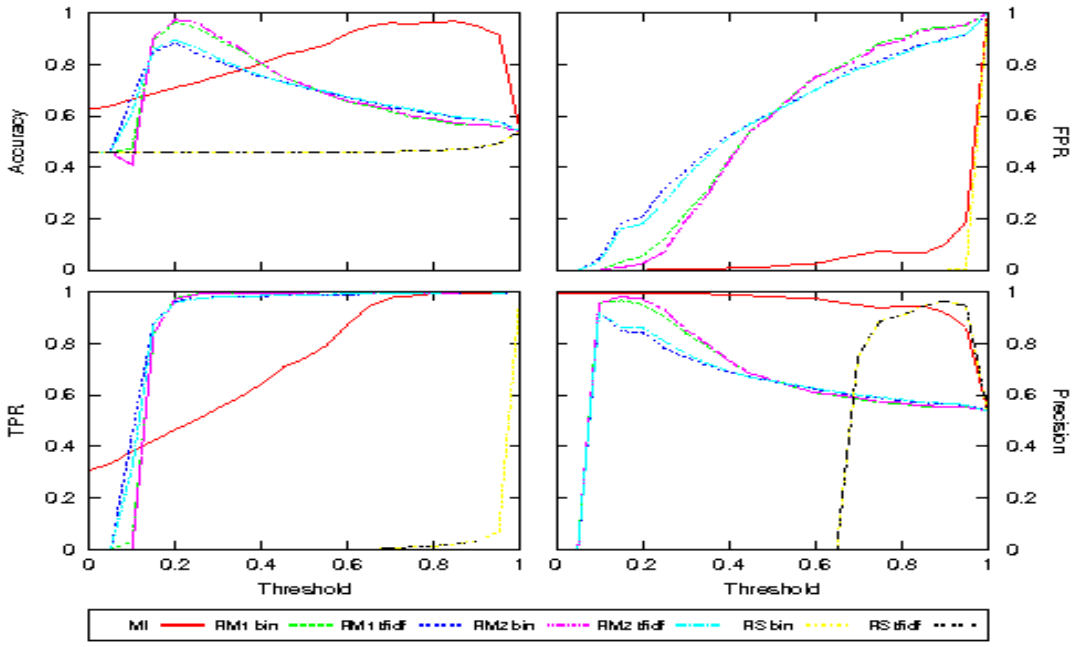

Figure B.25

5-gram, Whole portion data set, 500-features

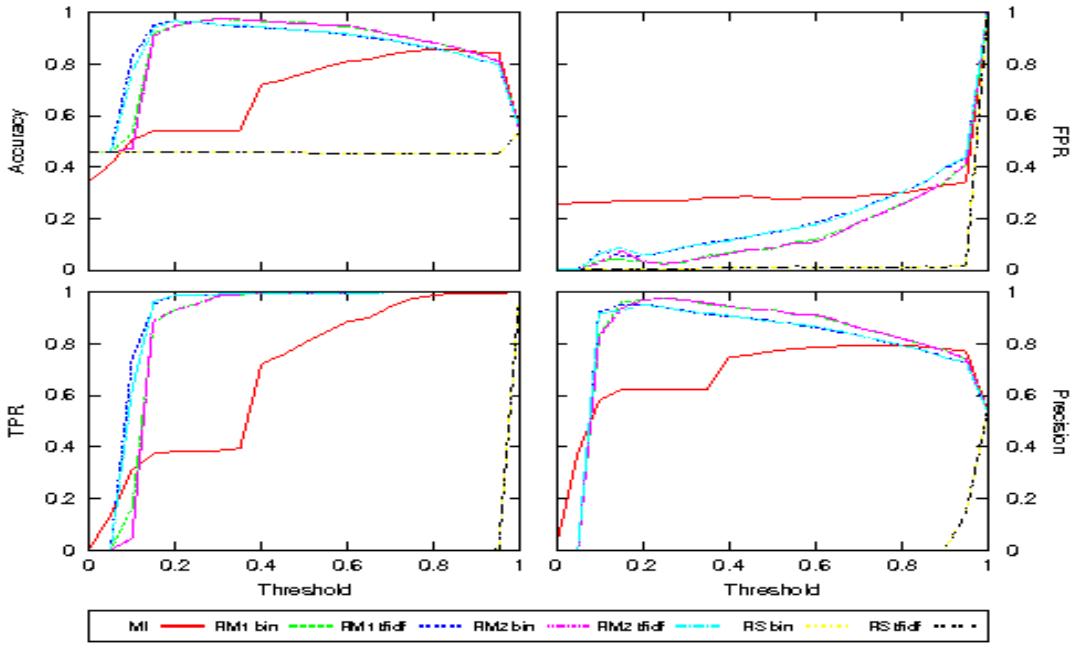

Figure B.26

5-gram, Data portion data set, 500-features

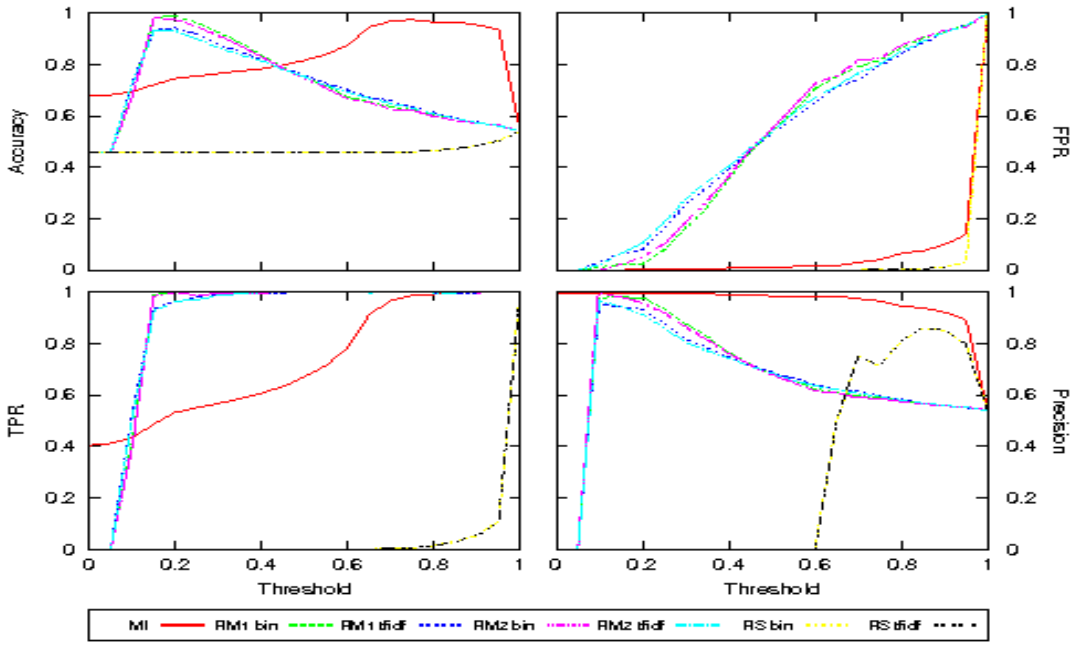

Figure B.27

5-gram, Code portion data set, 500-features

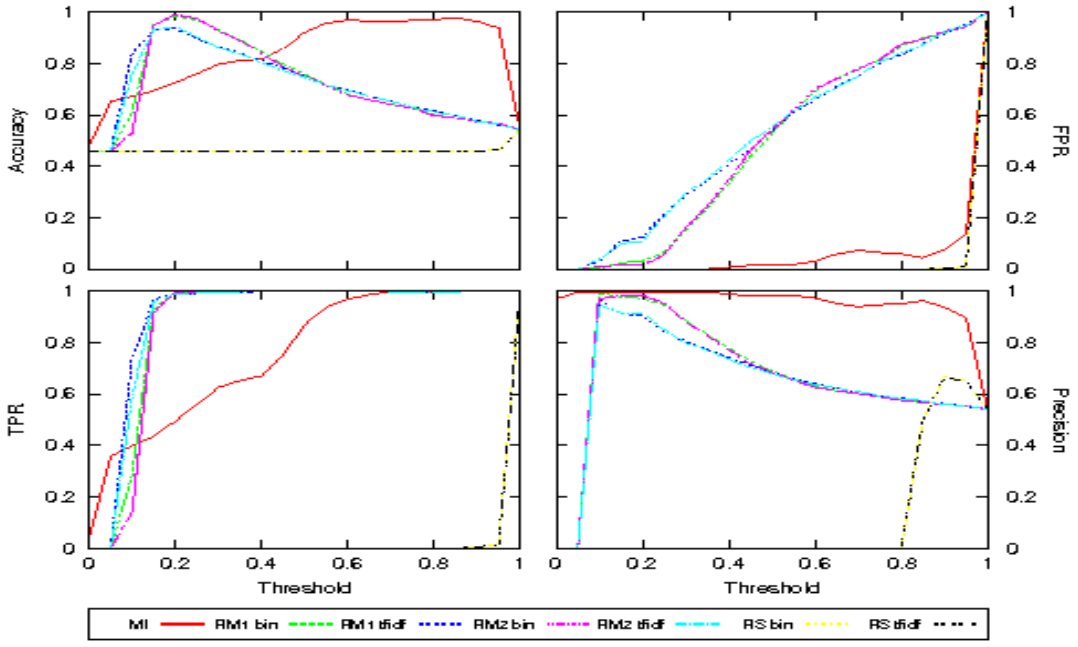

Figure B.28

5-gram, Combination portion data set, 500-features

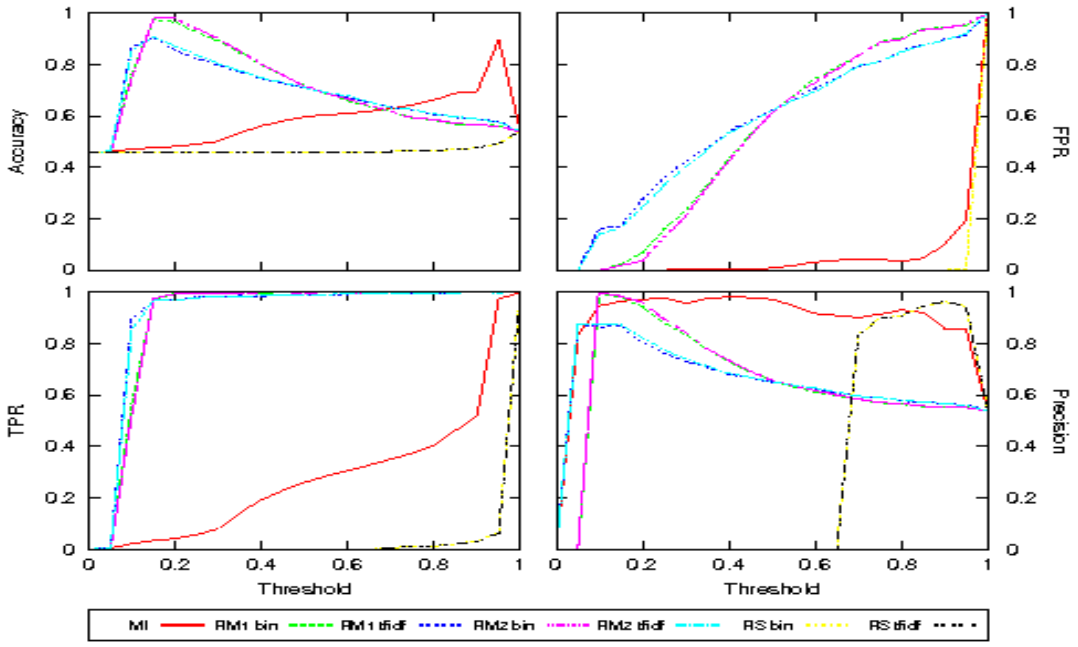

Figure B.29

5-gram, Whole portion data set, 1000-features

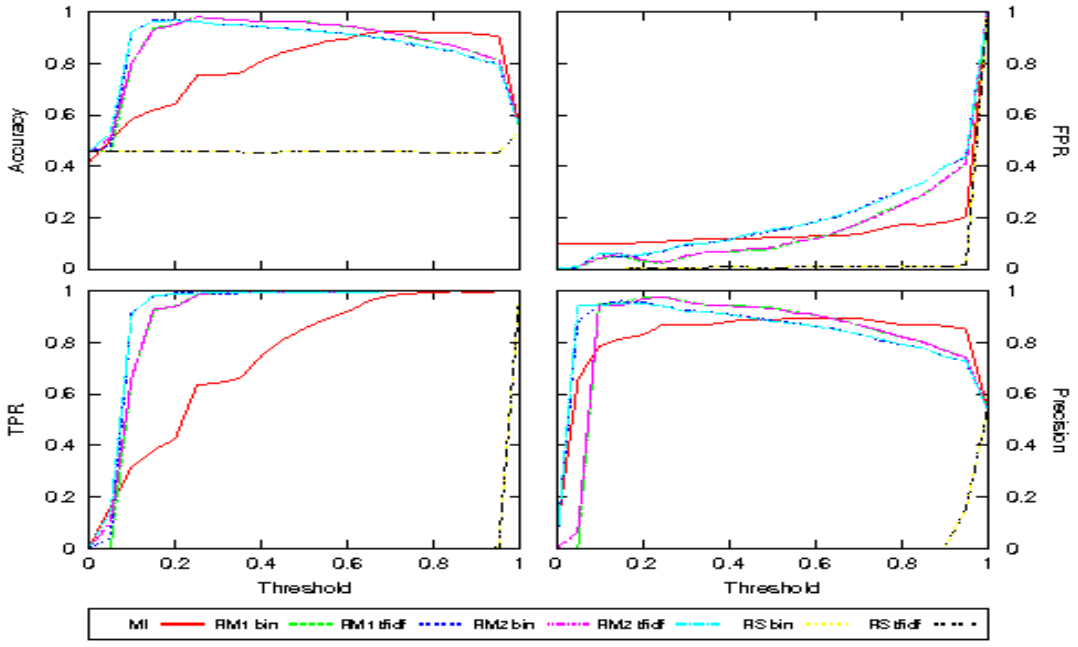

Figure B.30

5-gram, Data portion data set, 1000-features

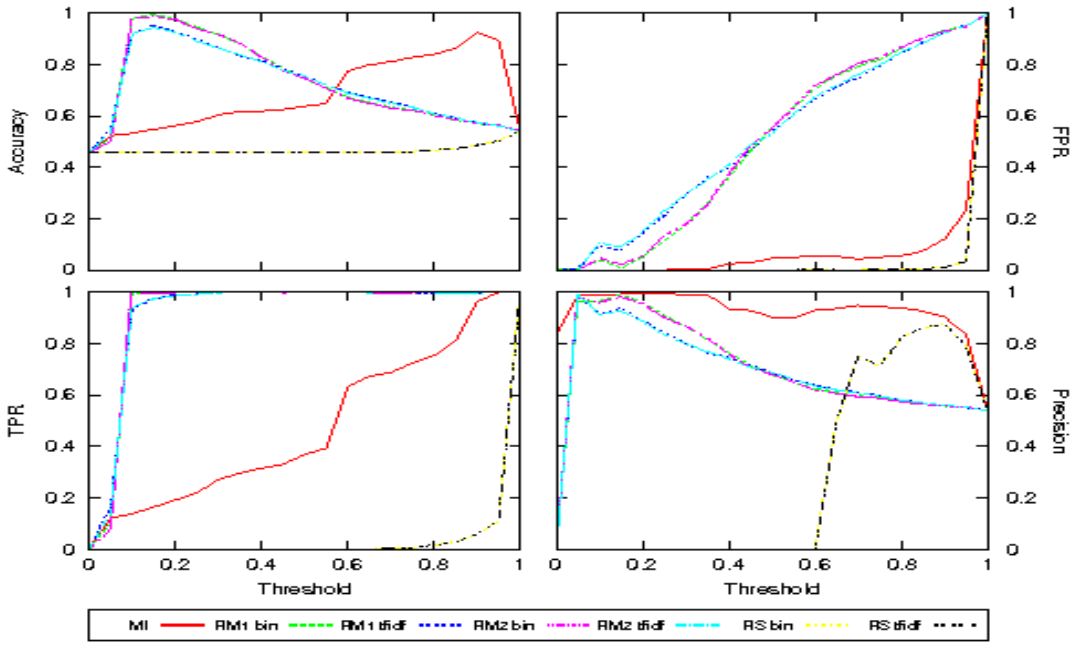

Figure B.31

5-gram, Code portion data set, 1000-features

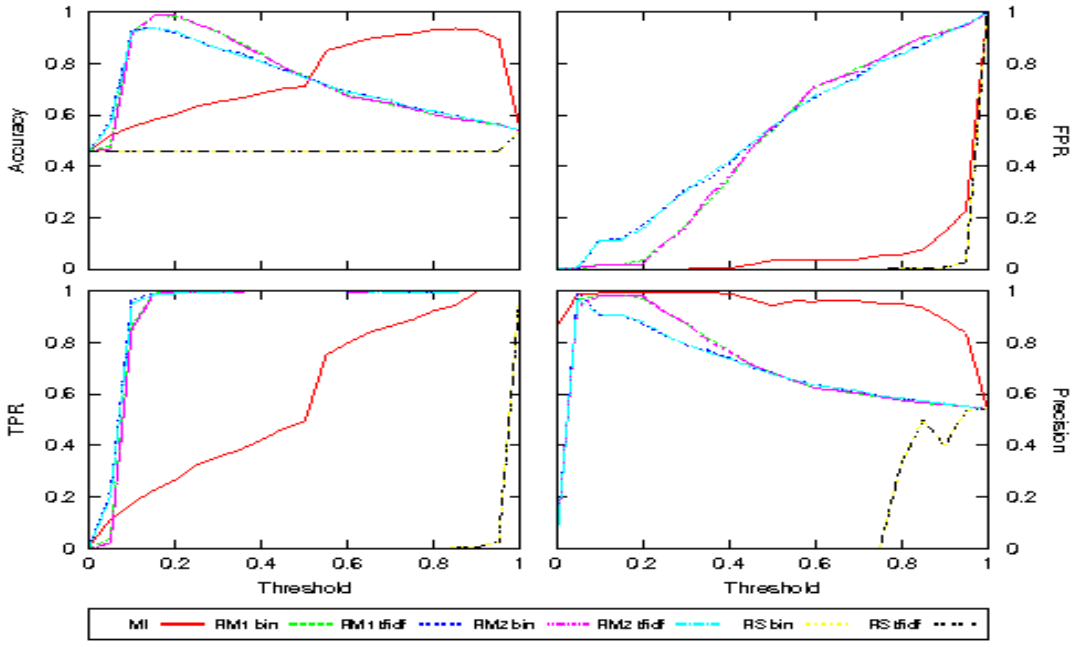

Figure B.32

5-gram, Combination portion data set, 1000-features

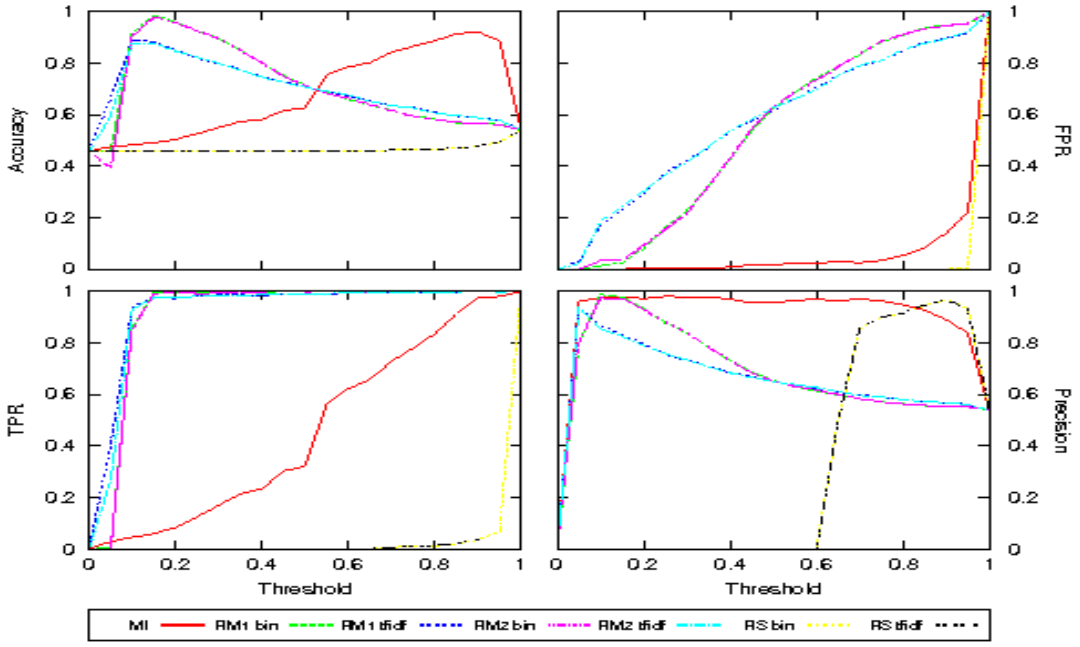

Figure B.33

5-gram, Whole portion data set, 1500-features

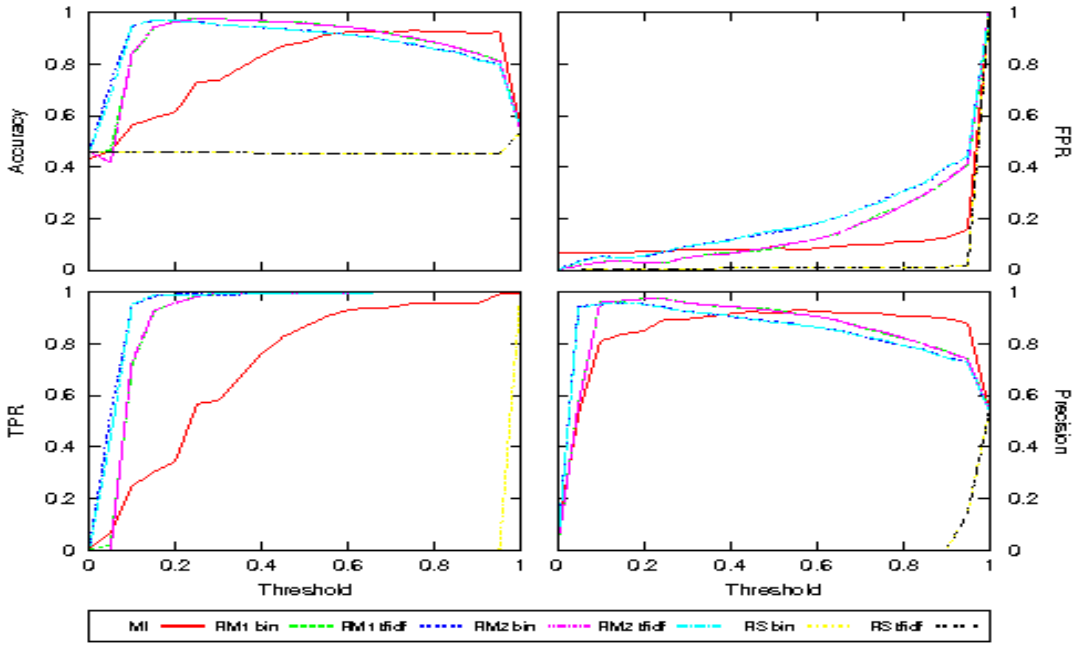

Figure B.34

5-gram, Data portion data set, 1500-features

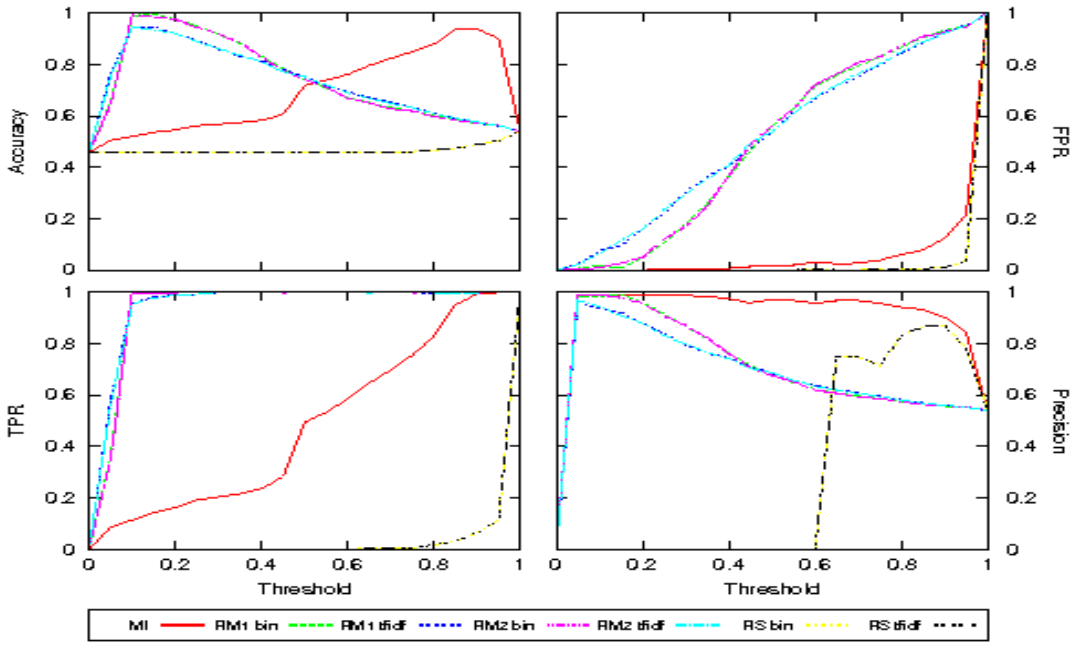

Figure B.35

5-gram, Code portion data set, 1500-features

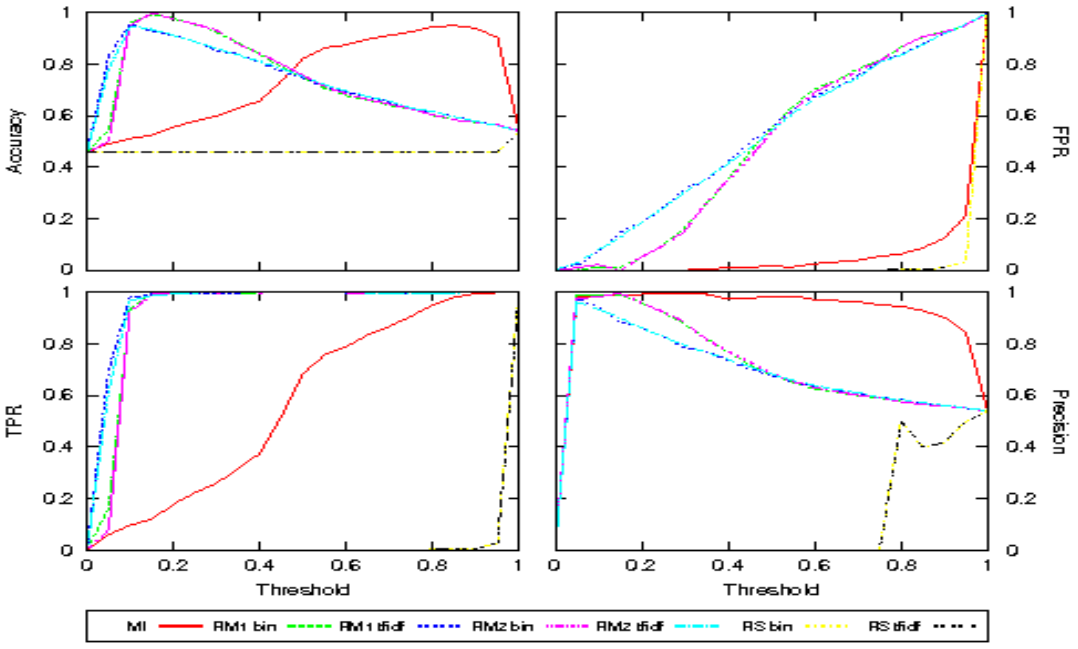

Figure B.36

5-gram, Combination portion data set, 1500-features

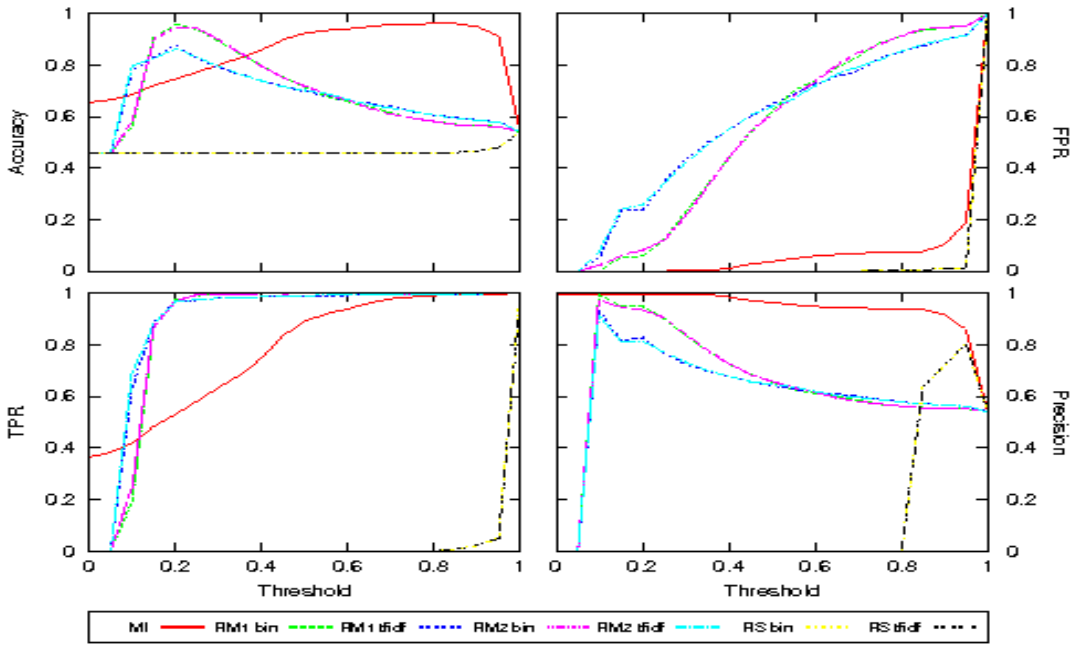

Figure B.37

6-gram, Whole portion data set, 500-features

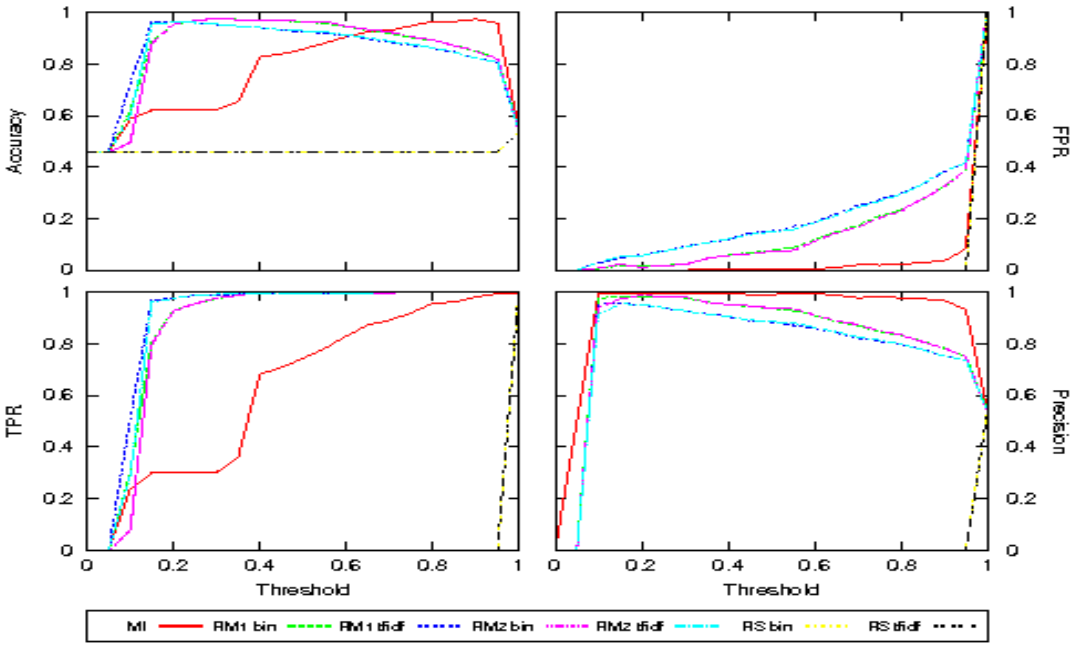

Figure B.38

6-gram, Data portion data set, 500-features

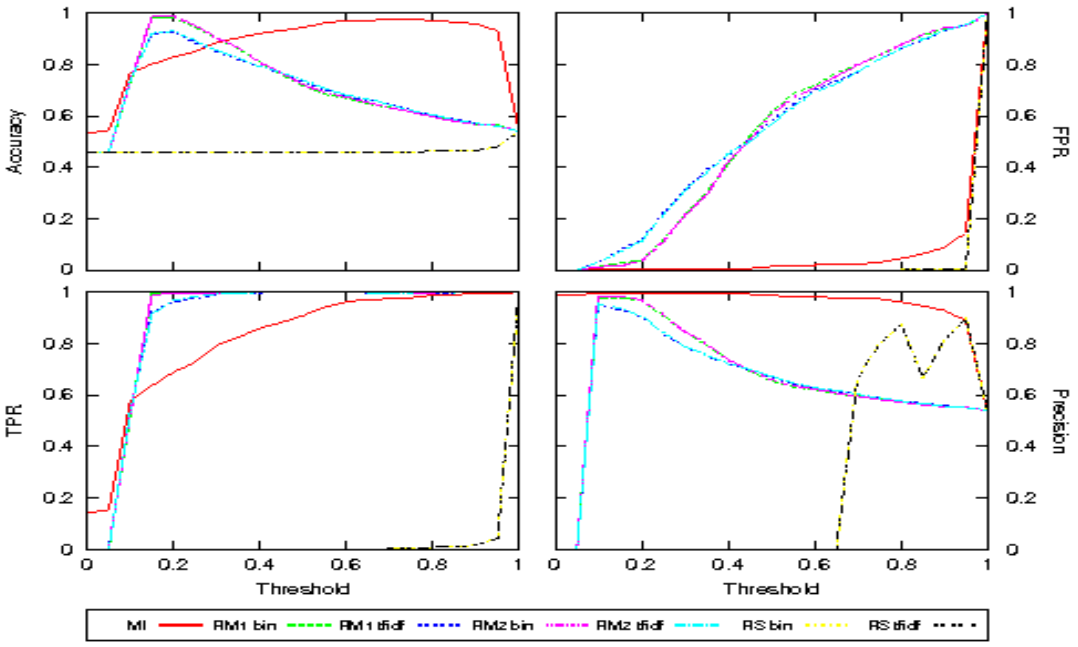

Figure B.39

6-gram, Code portion data set, 500-features

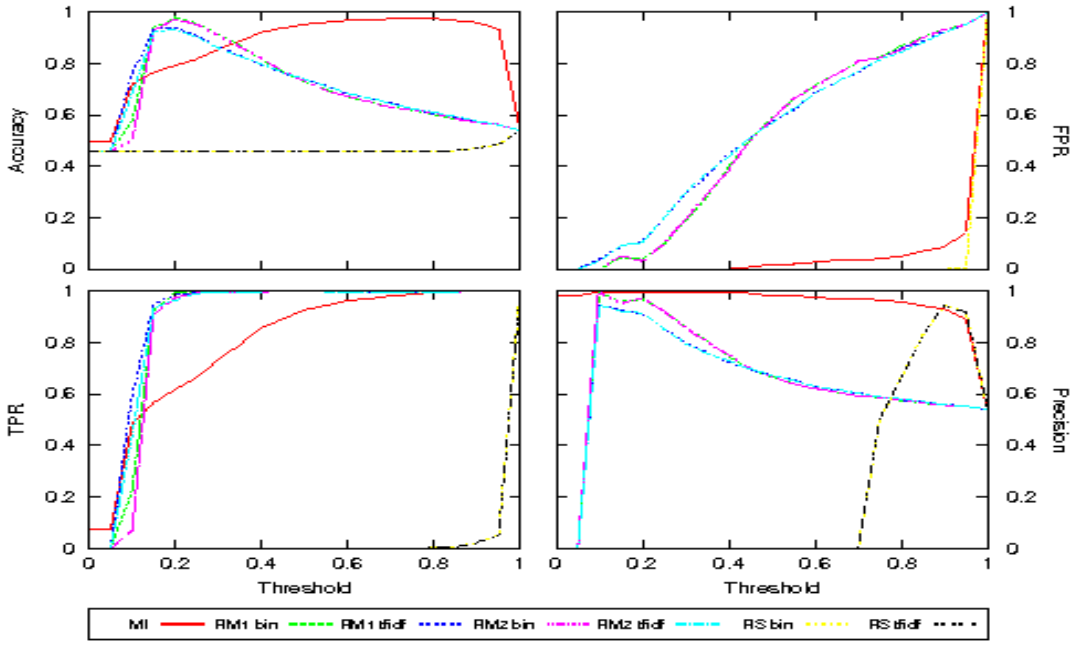

Figure B.40

6-gram, Combination portion data set, 500-features

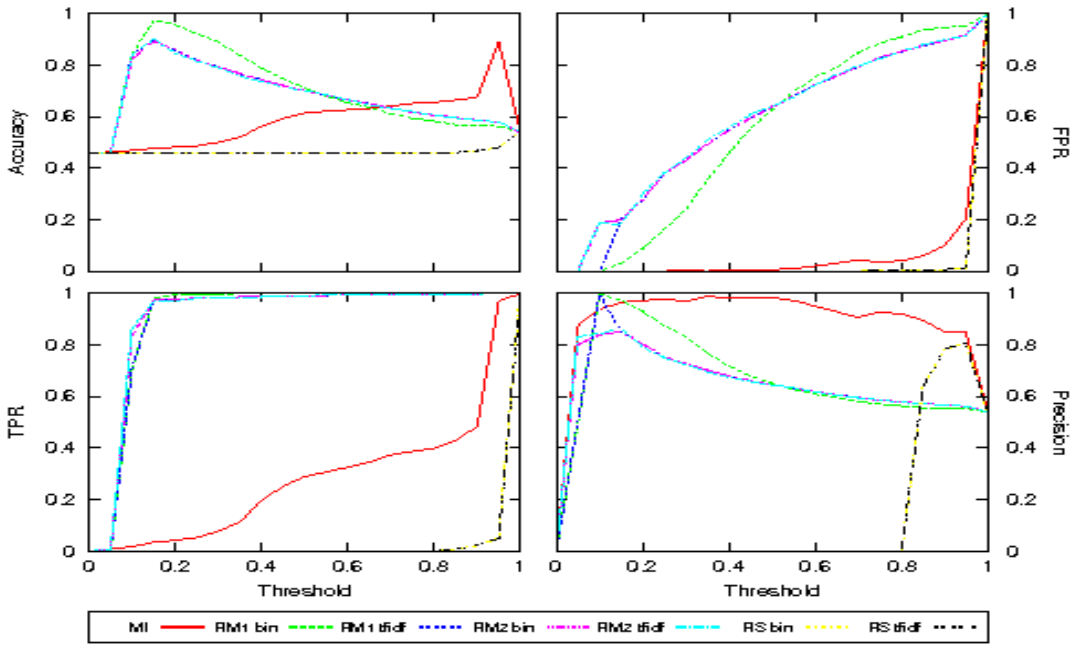

Figure B.41

6-gram, Whole portion data set, 1000-features

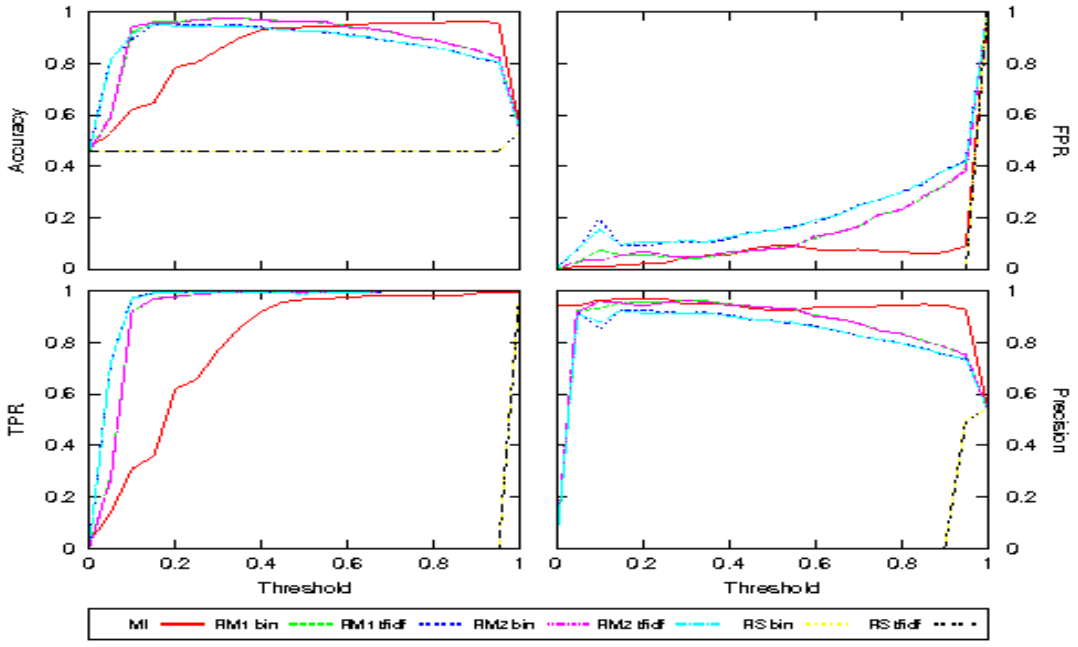

Figure B.42

6-gram, Data portion data set, 1000-features

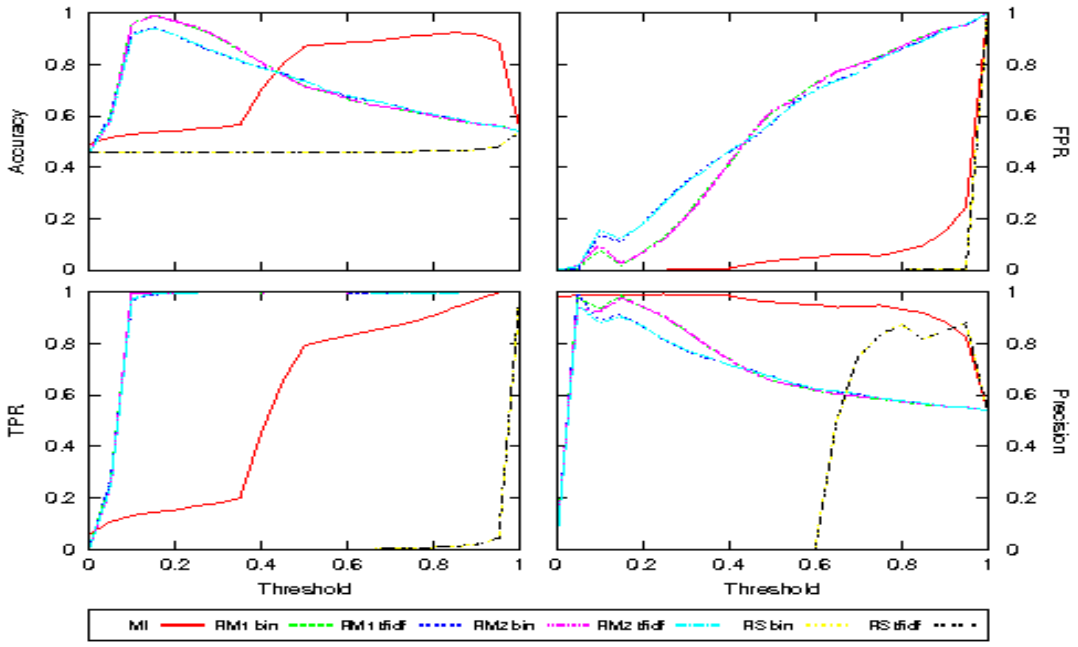

Figure B.43

6-gram, Code portion data set, 1000-features

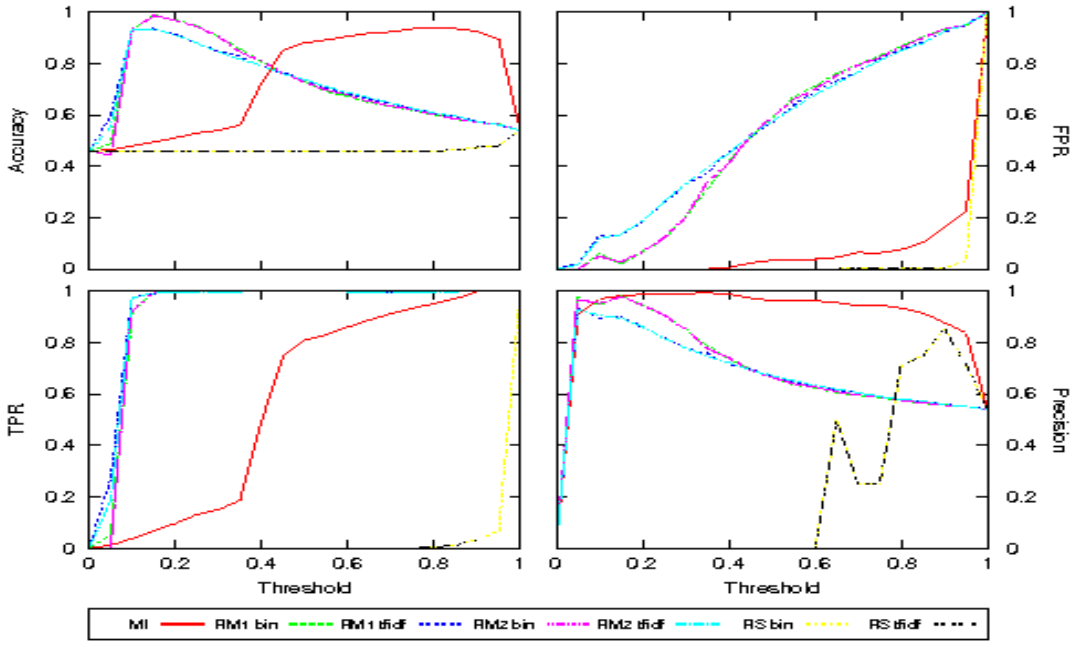

Figure B.44

6-gram, Combination portion data set, 1000-features

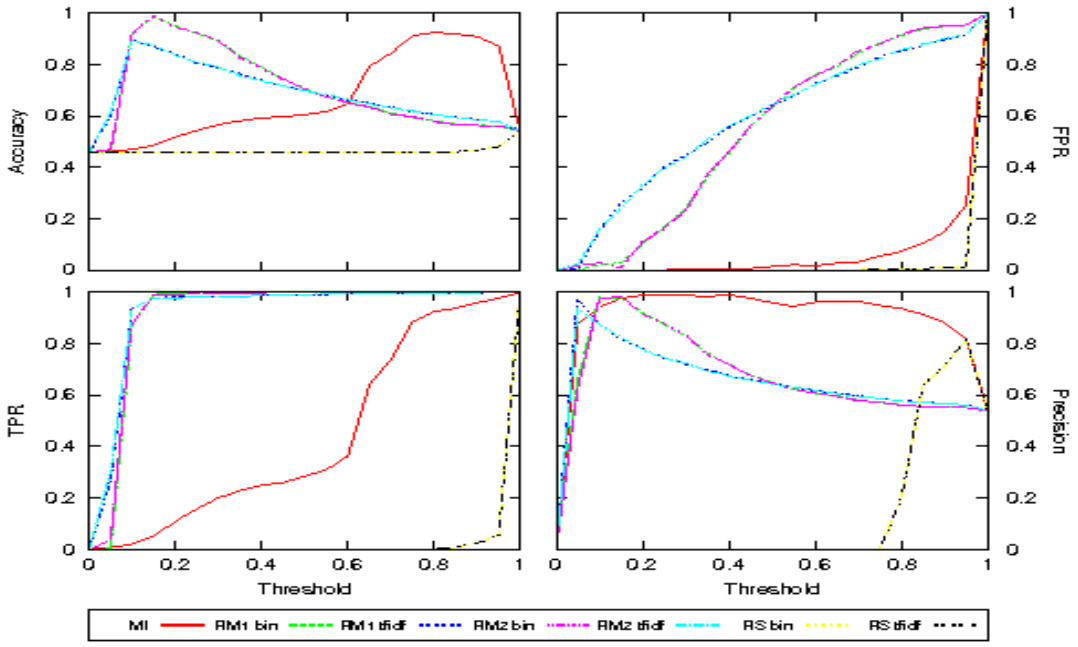

Figure B.45

6-gram, Whole portion data set, 1500-features

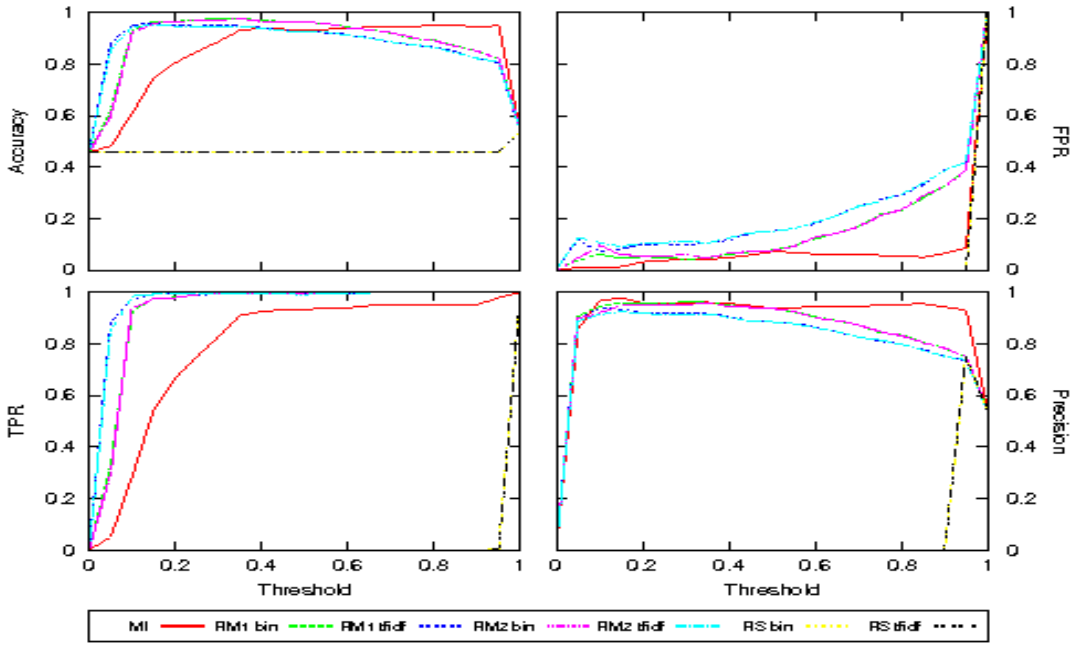

Figure B.46

6-gram, Data portion data set, 1500-features

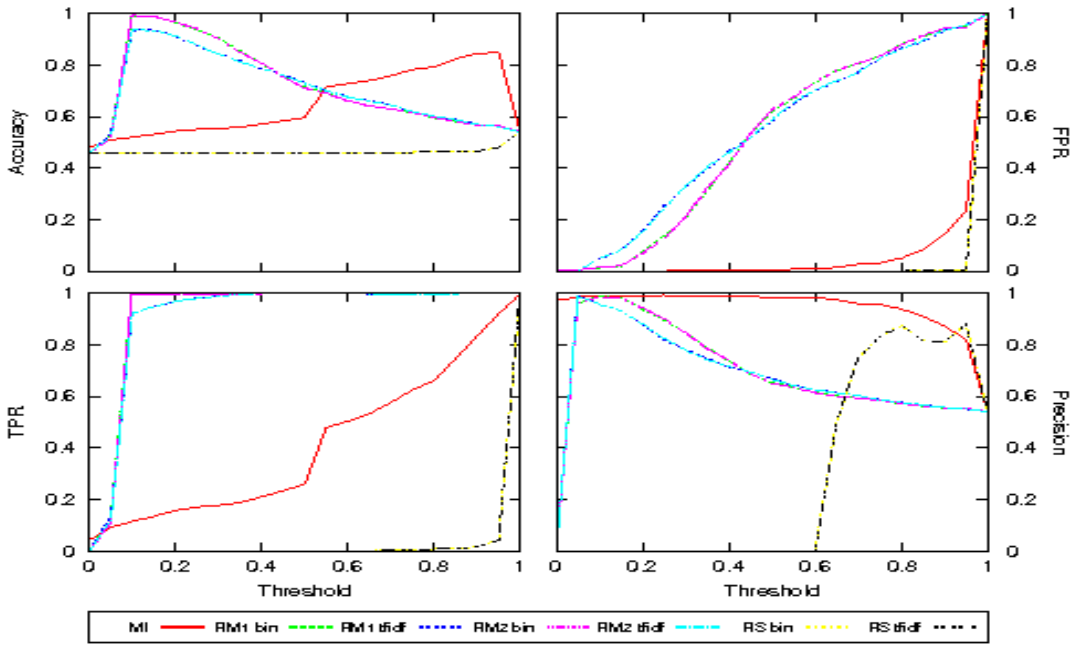

Figure B.47

6-gram, Code portion data set, 1500-features

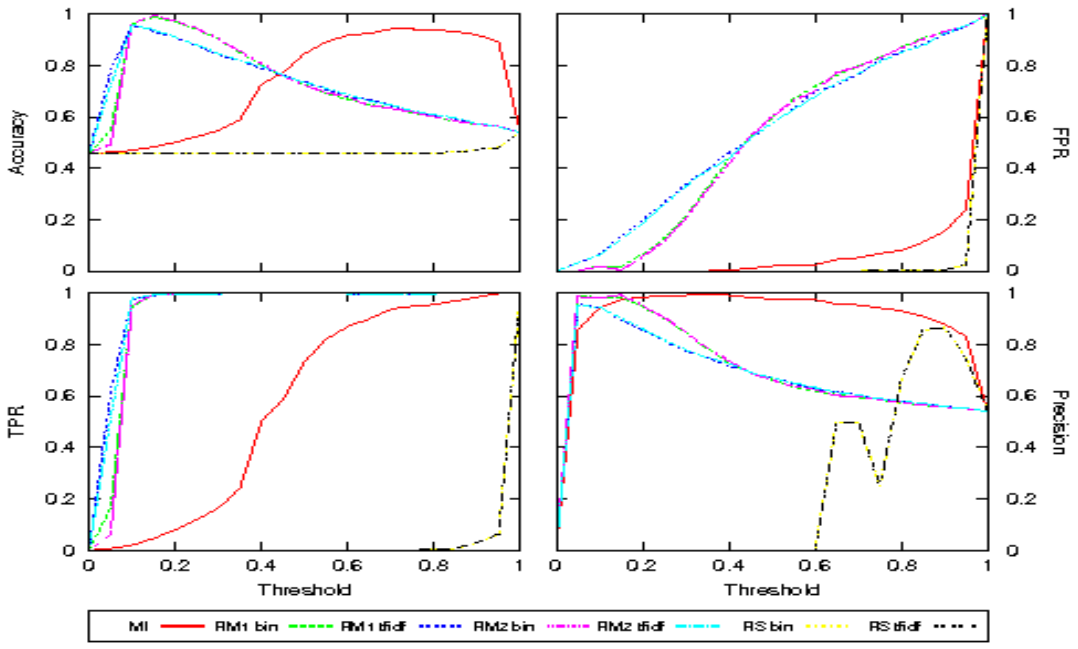

Figure B.48

6-gram, Combination portion data set, 1500-features

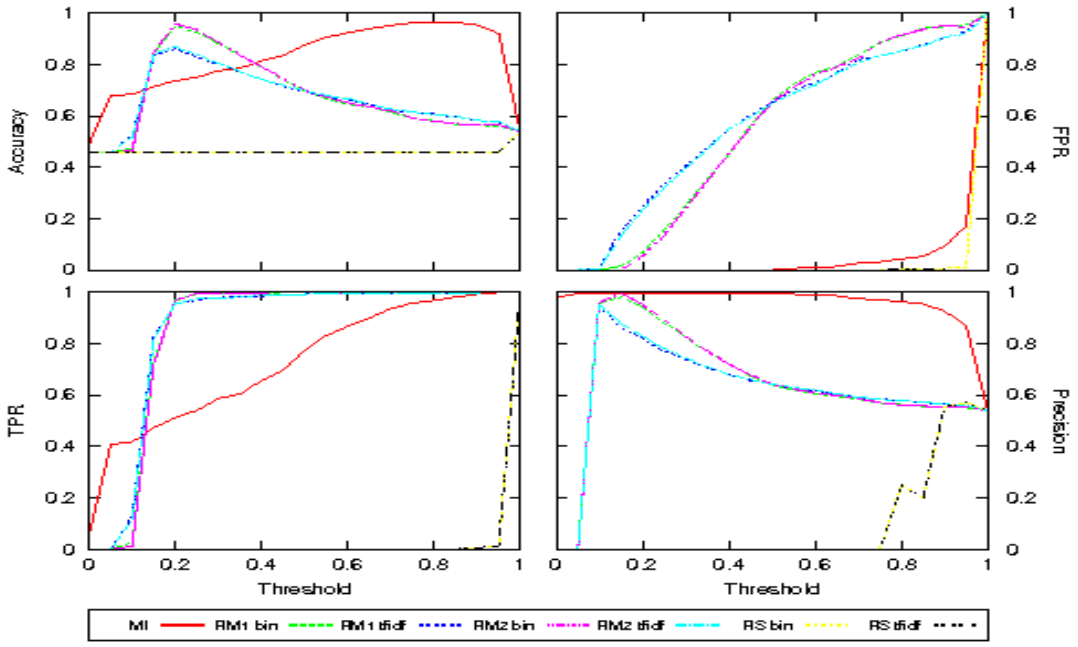

Figure B.49

7-gram, Whole portion data set, 500-features

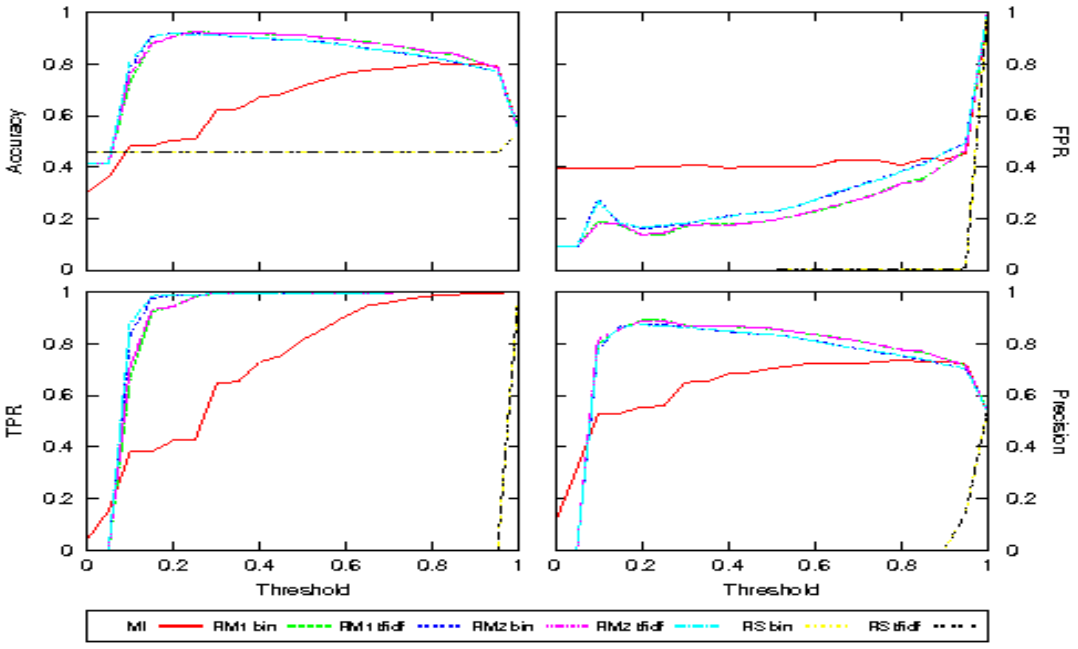

Figure B.50

7-gram, Data portion data set, 500-features

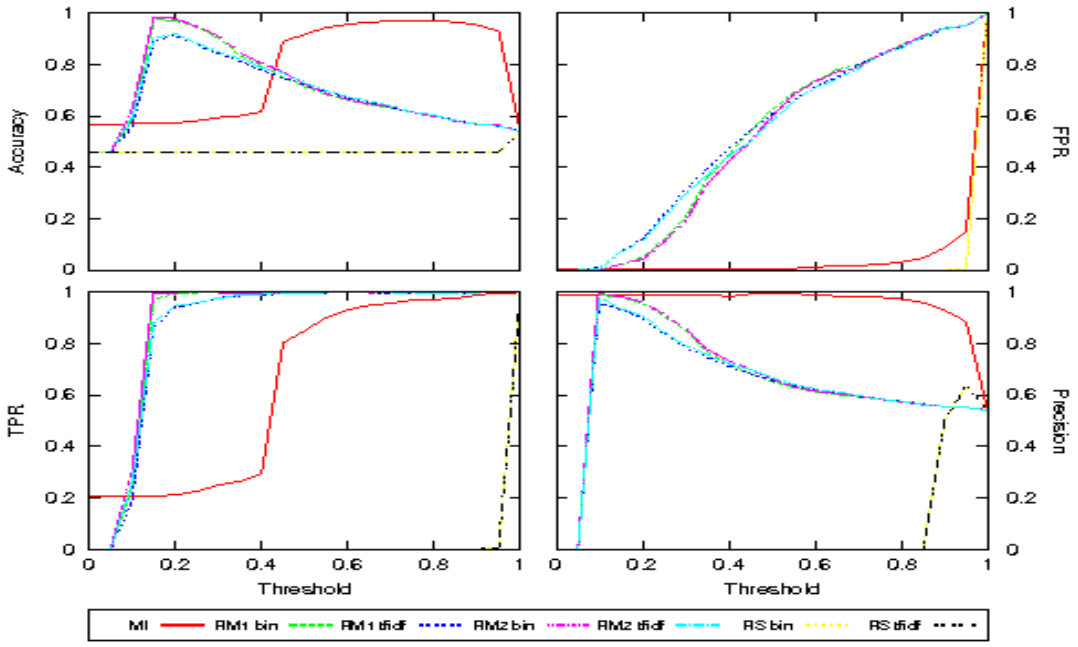

Figure B.51

7-gram, Code portion data set, 500-features

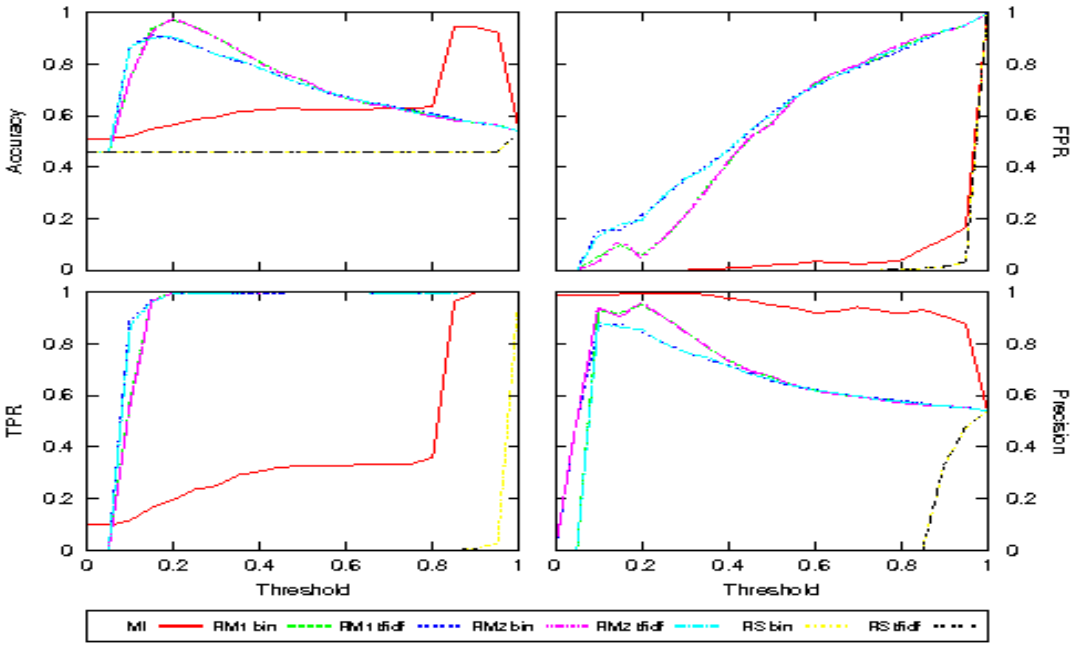

Figure B.52

7-gram, Combination portion data set, 500-features

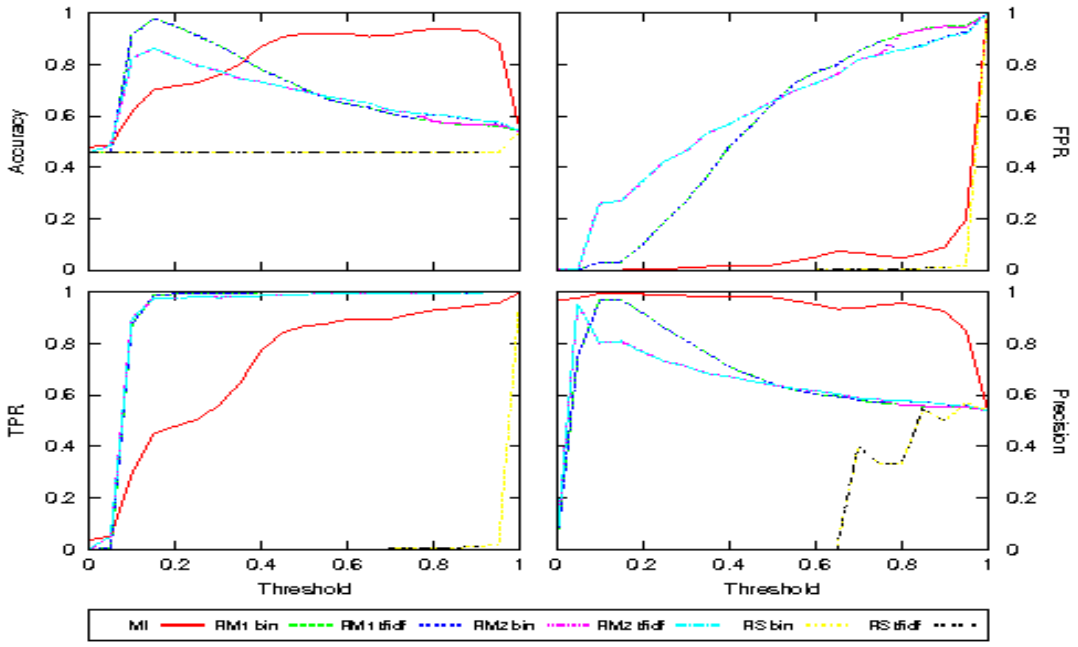

Figure B.53

7-gram, Whole portion data set, 1000-features

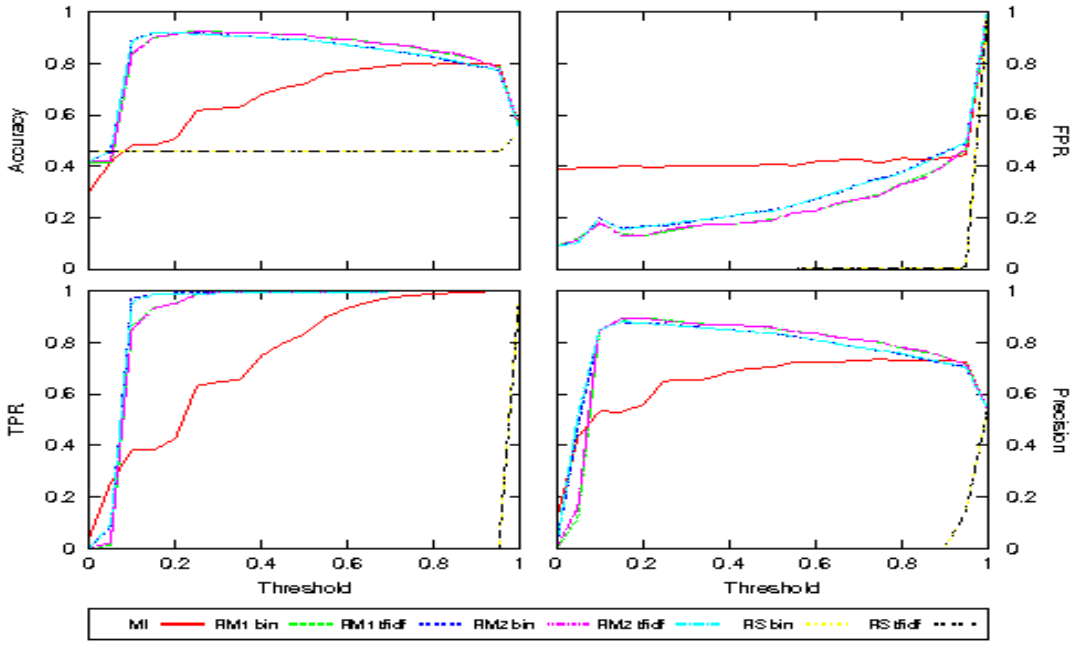

Figure B.54

7-gram, Data portion data set, 1000-features

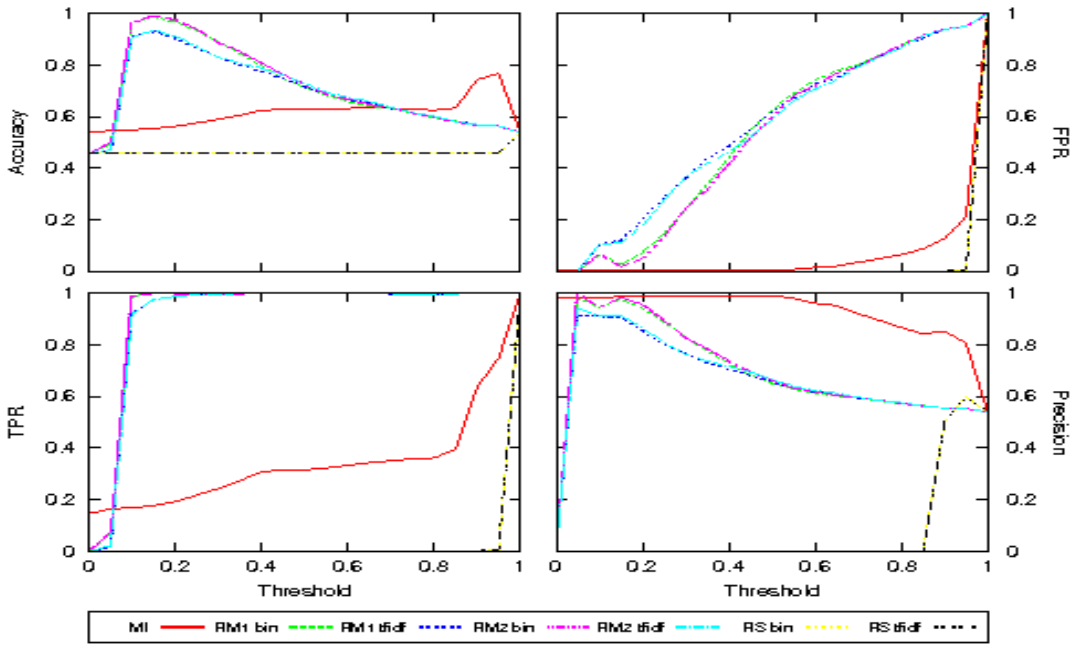

Figure B.55

7-gram, Code portion data set, 1000-features

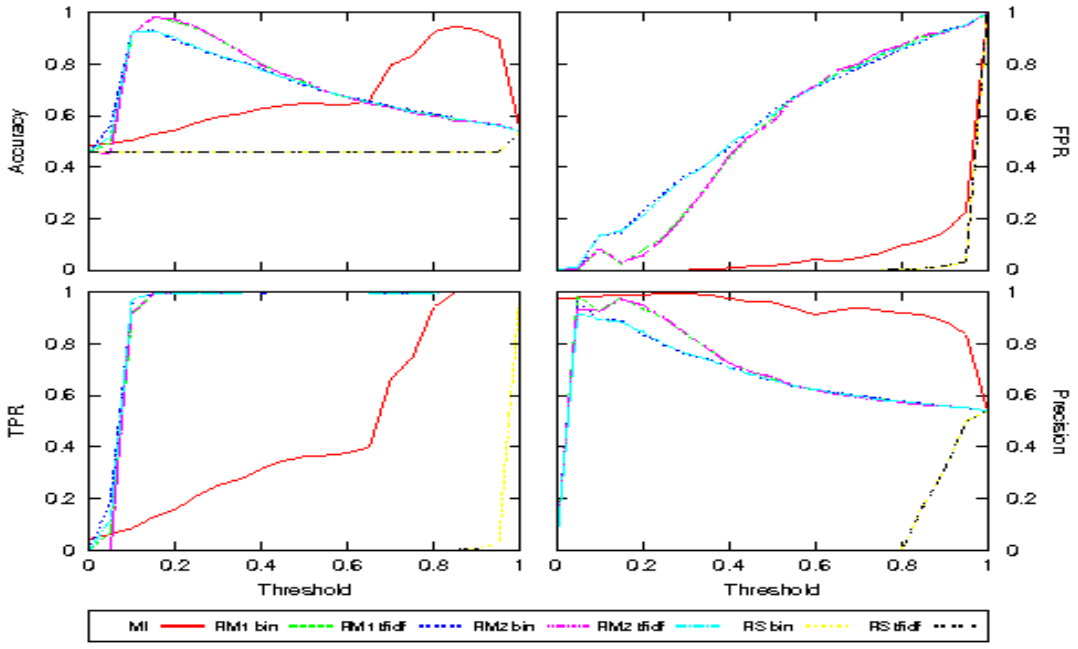

Figure B.56

7-gram, Combination portion data set, 1000-features

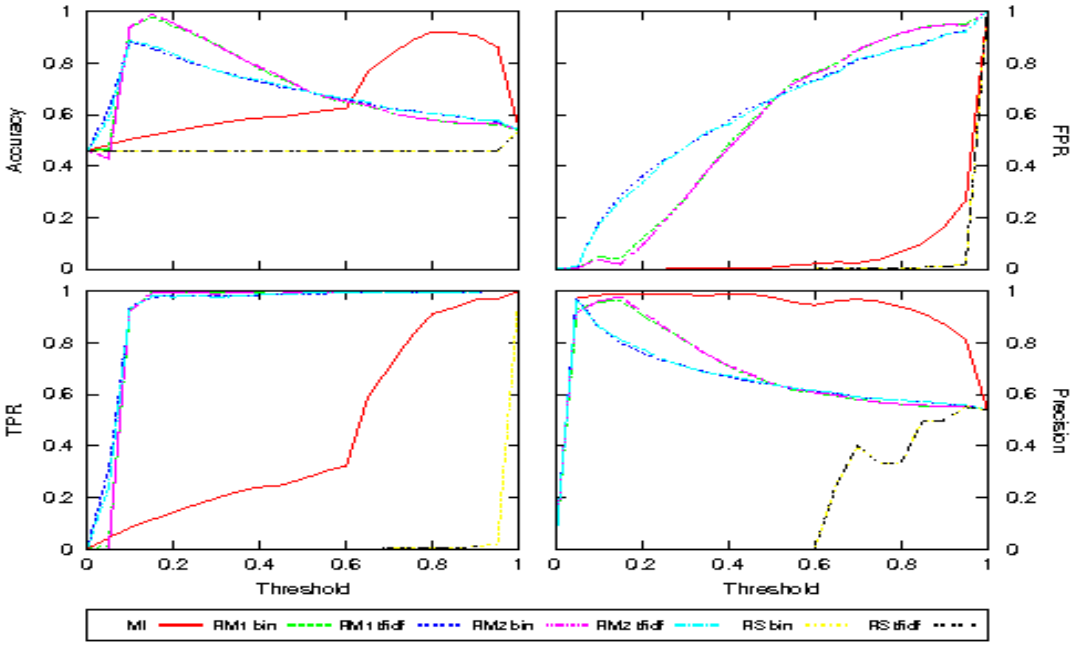

Figure B.57

7-gram, Whole portion data set, 1500-features

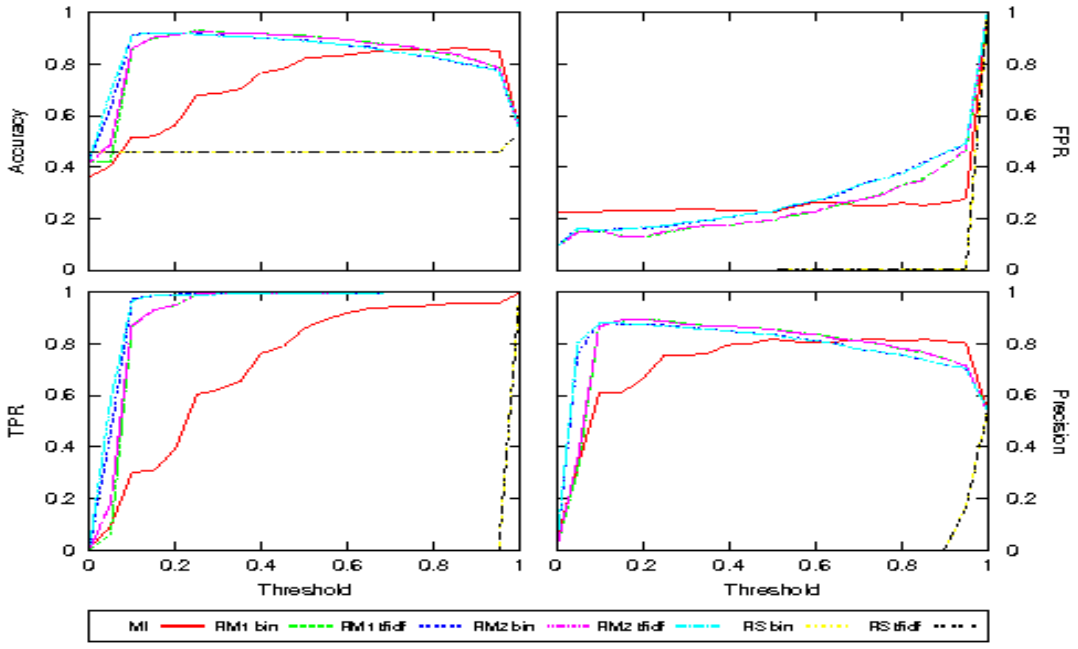

Figure B.58

7-gram, Data portion data set, 1500-features

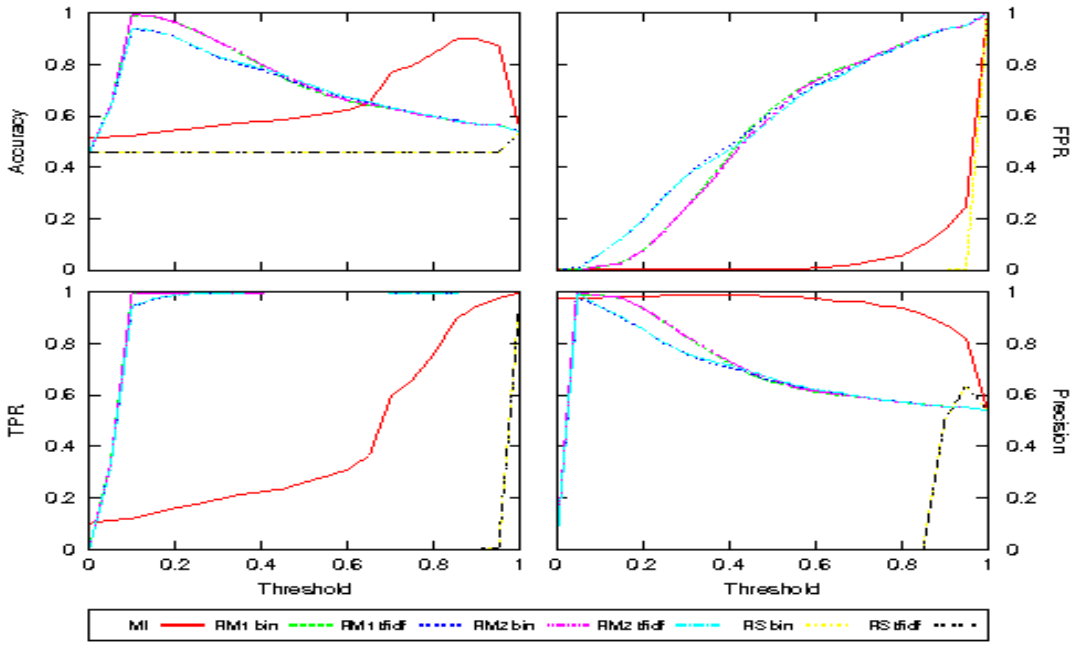

Figure B.59

7-gram, Code portion data set, 1500-features

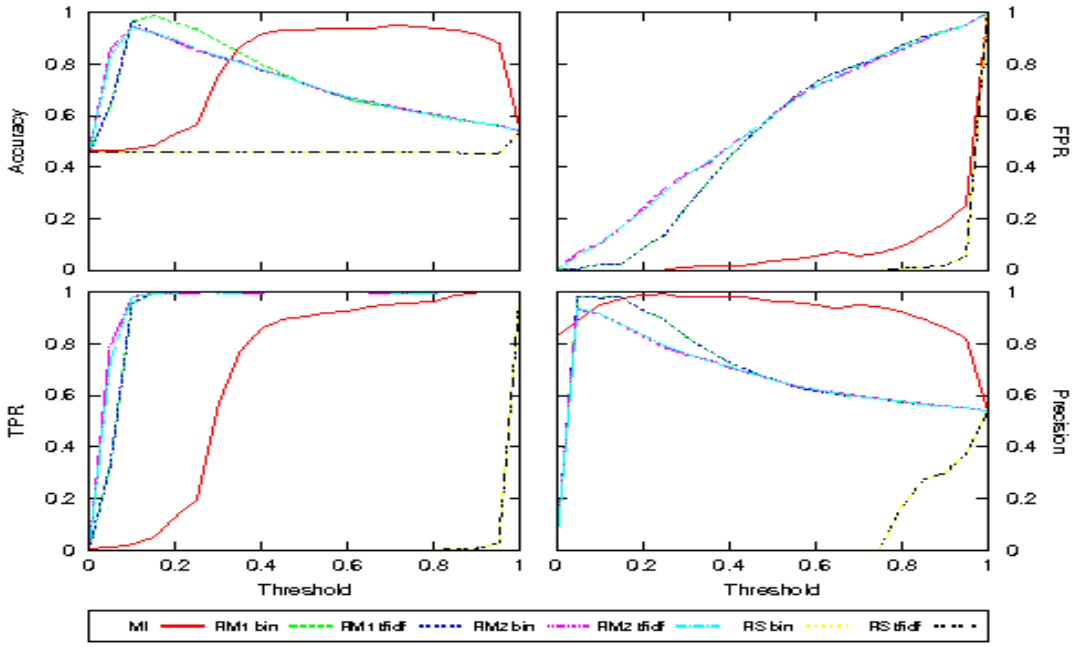

Figure B.60

7-gram, Combination portion data set, 1500-features