Use of K-Nearest Neighbor classifier for intrusion detection

Yihua Liao, V. Rao Vemuri

Department of Computer Science University of California, Davis One Shields Avenue, Davis, CA 95616, USA {yhliao, rvemuri}@ucdavis.edu

Abstract

A new approach, based on the k-Nearest Neighbor (kNN) classifier, is used to classify program behavior as normal or intrusive. Program behavior, in turn, is represented by frequencies of system calls. Each system call is treated as a word and the collection of system calls over each program execution as a document. These documents are then classified using kNN classifier, a popular method in text categorization. This method seems to offer some computational advantages over those that seek to characterize program behavior with short sequences of system calls and generate individual program profiles. Preliminary experiments with 1998 DARPA BSM audit data show that the kNN classifier can effectively detect intrusive attacks and achieve a low false positive rate.

Key words: k-Nearest Neighbor classifier, intrusion detection, system calls, text categorization, program profile.

1. Introduction

Computer security vulnerabilities will always exist as long as we have flawed security policies, poorly configured computer systems, or imperfect software programs. Intrusion detection systems play an important role in detecting attacks that exploit these vulnerabilities or flaws in computer systems [1]. An ideal intrusion detection system is the one that has 100% attack detection rate along with 0% false positive rate (the rate of mis-classified normal behavior), requires light load of monitoring, and involves minor calculation or overhead. Current intrusion detection systems, however, are plagued by either high false alarm probability or low attack detection accuracy.

An earlier version of this paper is to appear in the Proceedings of the 11th USENIX Security Symposium, San Francisco, CA, August 2002

There are two general approaches to intrusion detection: misuse detection and anomaly detection. Misuse detection via signature verification compares a user's activities with the known signatures of attackers attempting to penetrate a system. While misuse detection is useful for finding known intrusion types, it can not detect novel attacks. Unlike misuse detection, anomaly detection identifies activities that deviate from established statistical patterns for users, systems or networks. Machine learning techniques have been used to capture the normal usage patterns and classify the new behavior as either normal or abnormal (for example, [2]). In spite of their capability of detecting unknown attacks, anomaly detection systems suffer from high false alarm rate when normal user profiles and system or network behavior vary widely. Anomaly detection can be combined with signature verification to detect attacks more efficiently.

In anomaly detection, there are many different levels on which an intrusion detection system could monitor activities on a computer system. The biggest challenge is to choose features that best characterize the user or system usage patterns so that non-intrusive activities would not be classified as anomalous. One could use, for instance, Unix Shell command lines, login events or system calls as observables to generate profiles of user behavior. Since a user's behavior can change frequently, user profiles have to be updated periodically to include the most recent changes. It is possible that a user can fool the system by slowly changing his or her profile and hiding malicious activities.

More recently, learning program behavior and building program profiles, especially those of

ELSEVIER ADVANCED TECHNOLOGY

Computers & Security
Vol 21, No 5, pp 439-448, 2002

0167-4048/02US$22.00

privileged programs, has become an alternative method in intrusion detection [3-7]. Intrusions most often occur when programs are misused. In Unix, intruders usually gain super-user status by exploiting privileged programs. A program profile can be generated by monitoring the program execution and capturing the system calls associated with the program. Compared to user behavior profiles, program profiles are more stable over time because the range of program behavior is more limited. Furthermore, it would be more difficult for attackers to perform intrusive activities without revealing their tracks in the execution logs. Therefore program profiles provide concise and stable tracks for intrusion detection. To date, however, almost all the research in learning program behavior has used short sequences of system calls as the observable, and generated a large individual database of system call sequences for each program. A program's normal behavior is characterized by its local ordering of system calls, and deviations from their local patterns are regarded as violations of an executing program. It is still a tedious and costly approach because system and application programs are constantly updated, and it is difficult to build profiles for all of them.

In this paper, we discuss a new technique for learning program behavior in intrusion detection. Our approach employs the *k*-Nearest Neighbor (*k*NN) classifier to categorize each new program behavior into either normal or intrusive class. The frequencies of system calls used by a program, instead of their local ordering, are used to characterize the program's behavior. Each system call is treated as a "word", and each process, i.e., program execution, as a "document". Then the *k*NN algorithm, which has been successful in text categorization applications [8], can be easily adapted to intrusion detection. Since there is no need to build a profile for each program and check every sequence during the new program execution, the amount of calculation involved is largely reduced.

The rest of this paper is organized as follows. Section 2 surveys related work. We explain the analogy between text categorization and intrusion detection and the *k*NN classifier in Section 3. Section 4 describes our experiments with the 1998 DARPA data, and Section 5 contains further discussions. Finally, we summarize our conclusions and future work in Section 6.

2. Related work

Ko et al. at UC Davis first proposed to specify the intended behavior of some privileged programs (setuid root programs and daemons in Unix) using a program policy specification language [9]. During the program execution, any violation of the specified behavior was considered "misuse". The main limitation of this method is the difficulty of determining the intended behavior and writing security specifications for all monitored programs. Nevertheless, this research opened the door of modeling program behavior for intrusion detection.

Forrest's group at the University of New Mexico introduced the idea of using short sequences of system calls issued by running programs as the discriminator for intrusion detection [3]. Normal behavior was defined in terms of short sequences of system calls of a certain length in a running Unix process, and a separate database of normal behavior was built for each process of interest. This work was extended with various classification schemes such as artificial immune systems [4], rule learning [5] and Hidden Markov Model (HMM) [6]. Ghosh and others [7] employed artificial neural network techniques to learn program behavior profiles with system call sequences for the 1998 DARPA BSM data. More than 150 program profiles were established. For each program, a neural network was trained and used for anomaly detection. Their Elman recurrent neural networks were

able to detect 77.3% of all intrusions with no false positives, and 100% of all attacks with about 10% mis-classified normal sessions.

Unlike most researchers who concentrated on building individual program profiles, Asaka et al. [10] introduced a method based on discriminant analysis. Without examining all system calls, an intrusion detection decision was made by analyzing only 11 system calls in a running program and calculating the program's Mahalanobis' distances to normal and intrusion groups ofthe training data. There were 4 instances that were misclassified out of 42 samples. Due to its small size of sample data, however, the feasibility of this approach still needs to be established.

Ye et al. attempted to compare the intrusion detection performance of methods that used system call frequencies and those that used the ordering of system calls [11]. The names of system calls were extracted from the audit data of both normal and intrusive runs, and labeled asnormal and intrusive respectively. It is our impression that they did not separate the system calls based on the programs executing. Since both the frequencies and the ordering of system calls areprogram dependent, this over simplification limits the impact of their work.

3. Methodology

This paper treats the system calls differently. Instead of looking at the local ordering of the system calls, our method uses the frequencies of system calls to characterize program behavior. Using the text processing metaphor, each system call is treated as a "word" in a long document and the set of system calls generated by a process is treated as the "document". This analogy makes it possible to bring the full spectrum of well-developed text processing methods [12] to bear on the intrusion detection problem. One such method is the *k*-nearest neighbor classification method. The details on text categorization and the *k*NN classifier are presented in the Appendix.

Table 1: Analogy between text categorization and intrusion detection when applying the kNN classifier.

Terms	Text categorization	Intrusion Detection
N	total number of documents	total number of processes
M	total number of distinct words	total number of distinct system calls
n_i	number of times *i*th word occurs	number of times *i*th system call was issued
f_{ij}	frequency of *i*th word in document *j*	frequency of *i*th system call in process *j*
D_j	*j*th training document	*j*th training process
X	test document	test process

Analogous to text categorization, each process is first represented as a vector, where each entry represents the occurrence of a system call during the process execution. Weighting techniques such as frequency weighting (see A.1) and *tf•idf* weighting (see A.2) are used to determine thevalues of vector entries. To categorize a new process into either normal or intrusive class, the *k*NNclassifier calculates the similarity between the new process and each training process instance (see A.3), and uses the class labels of the *k* most similar neighbors to predict the class of the new process.The underlying assumption is that processes belonging to the same class will cluster together in the vector space. Table 1 illustrates the similarity in some respects between text categorizationand intrusion detection when applying the *k*NN classifier.

There are some advantages to applying text categorization methods to intrusion detection. First and foremost, the size of the system-call vocabulary is very limited. There are less than 100 distinct system calls in the DARPA BSM data, while a typical text categorization problem could have over 15000 unique words [12]. Thus the dimension of process vectors is significantly reduced, and it is not necessary to apply any dimensionality reduction techniques. Second, we can consider intrusion detection as a binary categorization problem, which makes adapting text categorization methods very straightforward.

4. Experiments

4.1 Data Set

We applied the *k*-Nearest Neighbor classifier to the 1998 DARPA data. The 1998 DARPA Intrusion Detection System Evaluation program provides a large sample of computer attacks embedded in normal background traffic [13]. The *TCPDUMP* and *BSM* audit data were collected on a network that simulated the network traffic of an Air Force Local Area Network. The audit logs contain seven weeks of training data and two weeks of testing data. There were 38 types ofnetwork-based attacks and several realistic intrusion scenarios conducted in the midst of normal background data.

We used the Basic Security Module (BSM) audit data collected from a victim Solaris machine inside the simulation network. The BSM audit logs contain information on system calls produced by programs running on the Solaris machine. See [14] for a detailed description of BSM events. We only recorded the names of system calls. Other attributes of BSM events, such as argumentsto the system call, object path and attribute, return value, etc., were not used here, although they could be valuable for other methods.

The DARPA data was labeled with session numbers. Each session corresponds to a TCP/IPconnection between two computers. Individual sessions can be programmatically extracted fromthe BSM audit data. Each session consists of one or more processes. A complete ordered list ofsystem calls is generated for every process. A sample system call list is shown below. The firstsystem call issued by Process 994 was *close*, *execve* was the next, then *open*, *mmap*, *open* and soon. The process ended with the system call *exit*.

Process ID: 994

close	*execve*	*open*	*mmap*	*open*	*mmap*	*mmap*	*munmap*	*mmap*
mmap	*close*	*open*	*mmap*	*close*	*open*	*mmap*	*mmap*	*munmap*
mmap	*close*	*close*	*munmap*	*open*	*ioctl*	*access*	*chown*	*ioctl*
access	*chmod*	*close*	*close*	*close*	*close*	*close*	*exit*	

The numbers of occurrences of individual system calls during the execution of a process were counted. Then text weighting techniques were used to transform the process into a vector. We used two weighting methods, frequency weighting defined by (A.1) and *tf•idf* weighting definedby (A.2), to encode the processes.

During our off-line data analysis, our data set included system calls executed by all processes except the processes of the Solaris operating system such as the inetd and shells, which usuallyspanned several audit log files.

4.2 Anomaly Detection

First we implemented intrusion detection solely based on normal program behavior. In order to ensure that all possible normal program behaviors are included, a large training data set is preferred for anomaly detection. On the other hand, a large training data set means large overhead in using a learning algorithm to model program behavior. There are 5 simulation days that were free of attacks during the seven-week training period. We arbitrarily picked 4 of them for training, and used the fifth one for testing. Our training normal data set consists of 606 distinct processesrunning on the

victim Solaris machine during these 4 simulation days. There are 50 distinct system calls observed from the training data set, which means each process is transformed into a vector of size 50. Table 2 lists all the 50 system calls.

Once we have the training data set for normal behavior, the *k*NN text categorization method can be easily adapted for anomaly detection. We scan the test audit data and extract the system call sequence for each new process. The new process is also transformed to a vector with the same weighting method. Then the similarity between the new process and each process in the training normal process data set is calculated using Equation (A.3). If the similarity score of one training normal process is equal to 1, which means the system call frequencies of the new process and thetraining process match perfectly, then the new process would be classified as a normal process immediately. Otherwise, the similarity scores are sorted and the *k* nearest neighbors are chosen to determine whether the new program execution is normal or not. We calculate the average similarity value of the *k* nearest neighbors (with highest similarity scores) and set a threshold. Only when the average similarity value is above the threshold, is the new process considered normal. The pseudo code for the adapted *k*NN algorithm is presented in Figure 1.

In intrusion detection, the Receiver Operating Characteristic (ROC) curve is usually used to measure the performance of the method. The ROC curve is a plot of intrusion detection accuracy against the false positive probability. It can be obtained by varying the detection threshold. We formed a test data set to evaluate the performance of the *k*NN classifier algorithm. The BSM data of the third day of the seventh training week was chosen as part of the test data set (none of the training processes was from this day). There was no attack launched on this day. It contains 412sessions and 5285 normal processes. (We did not require the test processes to be distinct in order to count false alarms for one day). The rest of the test data set consists of 55 intrusive sessions chosen from the seven-week DARPA training data. There are 35 clear or stealthy attack instances included in these intrusive sessions (some attacks involve multiple sessions), representing all types of attacks and intrusion scenarios in the seven-week training data. Stealthy attacks attempt to hide perpetrator's actions from someone who is monitoring the system, or the intrusion detectionsystem. Some duplicate attack sessions of the types *eject* and *warezclient* were skipped and not included in the test data set. When a process is categorized as abnormal, the session that the process is associated with is classified as an attack session. The intrusion detection accuracy iscalculated as the rate of detected attacks. Each attacks counts as one detection, even with multiple sessions. Unlike the groups who participated in the 1998 DARPA Intrusion Detection Evaluation

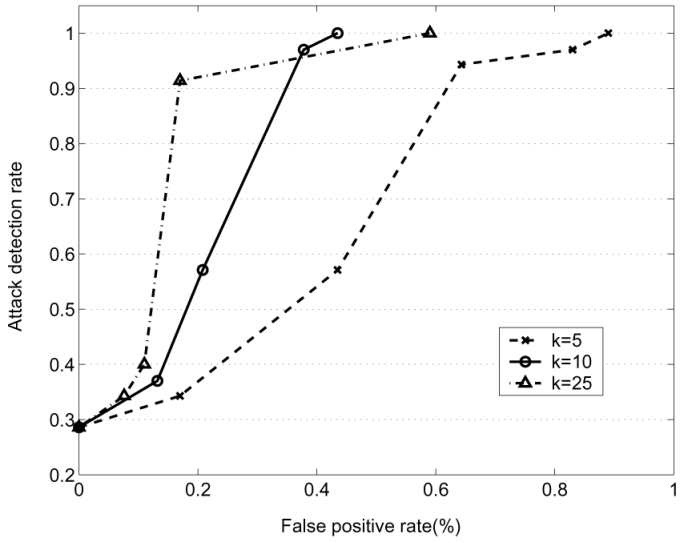
Table 2: List of 50 distinct system calls that appear in the training data set.

access	audit	auditon	chdir	chmod	chown	close	creat
execve	exit	fchdir	fchown	fcntl	fork	fork1	getaudit
getmsg	ioctl	kill	link	login	logout	lstat	memcntl
mkdir	mmap	munmap	nice	open	pathdonf	pipe	putmsg
readlink	rename	rmdir	setaudit	setegid	seteuid	setgid	setgroups
setpgrp	setrlimit	setuid	stat	statvfs	su	sysinfo	unlink
utime	vfork						

Figure 1: Pseudo code for the kNN classifier algorithm for anomaly detection.

build the training normal data set D;
for each process X in the test data **do**
  if X has an unknown system call **then**
    X is abnormal;
  else then
    for each process D_j in training data **do**
      calculate $sim(X, D_j)$;
      if $sim(X, D_j)$ equals 1.0 **then**
        X is normal; exit;
    find k biggest scores of $sim(X, D)$;
    calculate sim_avg for k-nearest neighbors;
    if sim_avg is greater than $threshold$ **then**
      X is normal;
    else then
      X is abnormal;

Figure 2: Performance of the kNN classifier method expressed in ROC curves for the tf•idf weighting method. False positive rate vs attack detection rate for k=5, 10 and 25.



program [15], we define our false positive probability as the rate of mis-classified processes, instead of mis-classified sessions.

The performance of the *k*NN classifier algorithm also depends on the value of *k*, the number of nearest neighbors of the test process. Usually the optimal value of *k* is empirically determined.We varied *k*'s value from 5 to 25. Figure 2 shows the ROC curves for 3 different *k* values when the processes are transformed with the *tf•idf* weighting method. For this particular data set, *k*=10 is a better choice than other values in that the attack detection rate reaches 100% faster. For *k*=10,the *k*NN classifier algorithm can detect 10 of the 35 attacks with zero false positive rate. And the detection rate reaches 100% rapidly when the threshold is raised to 0.72 and the false positive rate remains as low as 0.44% (23 false alarms out of 5285 normal processes) for the whole simulation day.

We also employed the frequency weighting method to transform the processes of the same training and test data sets. Similarly, for the frequency weighting method, *k*=15 provides the lowest false positive rate 0.87% (46 false alarms out of 5285 normal processes) when the attack detection rate reaches 100% with the threshold value of 0.99. The reason for the high threshold value is that some attack instances are very similar to the normal processes with the frequency weight. A comparison of two different weighting methods is shown in Figure 3. while the frequency weighting method offers a desirable high attack detection rate (86%) at zero false positives,the *tf•idf* weighting method provides lower false positive rate at 100% attack detection rate. It appears that the *tf•idf* weighting can make process vectors of two classes more distinguishable than the frequency weighting. Therefore, a lower threshold value is needed, and better false positive rate can be achieved with the *tf•idf* weighting method.

4.3 Anomaly Detection Combined with Signature Verification

We have just shown that the *k*NN classifier algorithm can be implemented for effective abnormality detection. The overall running time of the *k*NN method is *O(N)*, where *N* is the number of processes in the training data set (usually *k* is a small constant). When *N* is large, this method could still be computationally expensive for some real-time intrusion detection systems. In order to detect attacks more effectively, the *k*NN anomaly detection can be easily integrated with signature verification. The malicious program behavior can be encoded into the training set of the classifier. After carefully studying the 35 attack instances within the seven-week DARPA training data, we generated a data set of 19 intrusive processes. This intrusion data set covers most attack types of the DARPA training data. It includes the most clearly malicious processes, including *ejectexploit*, *formatexploit*, *ffbexploit* [16] and so on.

For the improved *k*NN algorithm, the training data set includes 606 normal processes as well as the 19 aforementioned intrusive processes. The 606 normal processes are the same as the ones in subsection 4.2. Each new test process is

compared to intrusive processes first. Whenever there is a perfect match, i.e., the cosine similarity is equal to 1.0, the new process is labeled as intrusive behavior (one could also check for near matches). Otherwise, the abnormal detection procedure in Figure 1 is performed. Due to the small amount of the intrusive processes in the training data set, this modification of the algorithm only causes minor additional calculation for normal testing processes.

The performance of the improved *k*NN classifier algorithm was evaluated with 24 attacks within the two-week DARPA testing audit data. The DARPA testing data contains some known attacks as well as novel ones. Some duplicate instances of the e*ject* attack were not included in the test data set. The false positive rate was evaluated with the same 5285 testing normal processes as described in Section 4.2. Table 3 presents the the attack detection accuracy for the *tf•idf* weighting (*k*=10 and *threshold* = 0.8) and the frequency weighting (*k*=15 and *threshold* = 0.99). For the *tf•idf* weighting method, the false positive rate is 0.59% (31 false alarms) when the threshold is adjustedto 0.8. For the frequency weighting, the false positive rate remains at 0.87% with the threshold of 0.99.

The two missed attack instances were a new denial of service attack, called *process table*.They matched with one of training normal processes exactly, which made it impossible for the *k*NN algorithm to detect. The *process table* attack was implemented by establishing connections to the telnet port of the victim machine every 4 seconds and exhausting its process table so that no new process could be launched [16]. Since this attack consists of abuse of a perfectly legal action, it didn't show any abnormality when we analyzed individual processes. Characterized by an unusually large number of connections active on a particular port, this denial of service attack, however, could be easily identified by other intrusion detection methods. Among the other 22 detected attacks, eight were captured with signature verification for the *tf•idf* weighting and two for the frequency weighting.

Figure 3: ROC curves for tf•idf weighting (k=10) and frequency weighting (k=15).

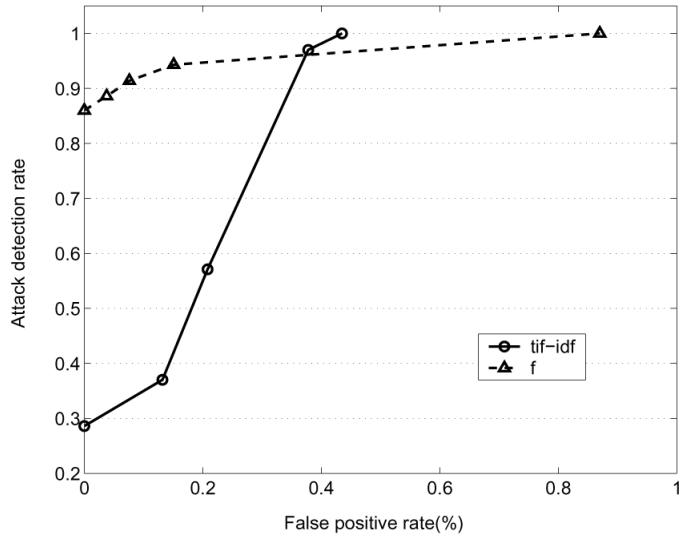


Table 3: Attack detection rate for DARPA testing data when anomaly detection is combined with signature verification.

Attack	Instances	Detected	Detection rate
Known attacks	16	16	100%
Novel attacks	8	6	75%
Total	24	22	91.7%

5. Discussion

The RSTCORP group [7] gave good performance during the evaluation of the 1998 DARPABSM data [15]. A neural network was trained for each program. Their Elman recurrent neural networks were able to detect 77.3% of all intrusions with no false positives, and 100% of all attacks with about 10% miss-classified normal sessions. which means 40 to 50 false positive alarms for a typical simulation day with 500 sessions. Their test data consisted of 139 normal sessions and 22 intrusive sessions. Since different test data sets were used, it is difficult to compare the performance of our *k*NN classifier with that of the Elman networks.

In spite of that, the *k*NN classifier avoids the time-consuming training process, and, more importantly, bypasses the need to learn individual program profiles separately. Thus the cost of learning program behavior is significantly decreased.

Unlike the *tf•idf* weighting, the frequency-weighting method assigns the number of occurrences of a system call during the process execution to a vector entry. Each process vector does not carry any information on other processes. A new training process could be easily added to the training data set without changing the weights of the existing training samples. Therefore the frequency-weighting method makes the *k*NN classifier method more suitable for dynamic environments that requires frequent updates of the training data.

In our current implementation, we used all the system calls to represent program behavior. The dimension of process vectors, and hence the classification cost, can be further reduced by only using the most influential system calls.

Our approach is predicated on the following properties: the frequencies of system calls issued by a program appear consistently across its normal executions and unseen system calls will beexecuted or unusual frequencies of the invoked system calls will appear when the program is exploited. We believe these properties hold true for many programs. However, if an intrusion does not reveal any anomaly in the frequencies of system calls, our method would miss it. For example, attacks that consist of abuse of perfectly normal processes such as *process table* would not be identified by the *k*NN classifier.

With the *k*NN classifier method, each process is classified when it terminates. We point out that it could still be suitable for real-time intrusion detection. Each intrusive attack is usually conducted within one or more sessions, and every session contains several processes. Since the *k*NN classifier method monitors the execution of each process, it is very likely that an attack can be detected while it is in operation. However, it is possible that an attacker can avoid being detected by not letting the process exit. Nonetheless, the *k*NN classifier could be integrated with other methods [18] that utilize the ordering information of system calls for performance enhancement.

6. Conclusion

In this paper we have proposed a new algorithm based on the *k*-Nearest Neighbor classifier method for modeling program behavior in intrusion detection. Our preliminary experiments withthe 1998 DARPA BSM audit data have shown that this approach is able to effectively detect intrusive program behavior. Compared to other methods using short system call sequences, the *k*NN classifier doesn't have to build separate profiles of short system call sequences for different programs, thus the calculation involved with classifying new program behavior is largely reduced. Our results also show that the a low false positive rate can be achieved. While this result may not hold against a more sophisticated data set, text categorization techniques appear to be well applicable to the domain of intrusion detection.

Further research is in progress to investigate the reliability and scaling properties of the *k*NN classifier method. In particular, we are studying the methods of selecting the most relevant system calls for classification, and conducting quantitative comparison between the *k*NN classifier andother machine learning methods. In addition, mixed modeling of program behavior using both local ordering of system calls and system call frequencies is another direction of our future work.

7. Acknowledgment

We thank Dr. Marc Zissman of Lincoln Laboratory at MIT for providing us the DARPA training and testing data. This work is

supported in part by the AFOSR grant F49620-01-1-0327 to the Center for Digital Security of the University of California, Davis.

Appendix

Text Categorization and K-Nearest Neighbor Classifier

Text categorization is the process of grouping text documents into one or more predefined categories based on their content. A number of statistical classification and machine learning techniques have been applied to text categorization, including regression models, Bayesian classifiers, decision trees, nearest neighbor classifiers, neural networks, and support vector machines [12].

The first step in text categorization is to transform documents, which typically are strings of characters, into a representation suitable for the learning algorithm and the classification task. The most commonly used document representation is the so-called vector space model. In this model, each document is represented by a vector of words. A word-by-document matrix **A** is used for a collection of documents, where each entry represents the occurrence of a word in a document, i.e., $\mathbf{A} = (a_{ij})$, where a_{ij} is the weight of word i in document j. There are several ways of determining the weight a_{ij}. Let f_{ij} be the frequency of word i in document j, N the number of documents in the collection, M the number of distinct words in the collection, and n_i the total number of times word i occurs in the whole collection. The simplest approach is Boolean weighting, which sets the weight a_{ij} to 1 if the word occurs in the document and 0 otherwise. Another simple approach, frequency weighting, uses the frequency of the word in the document:

$$a_{ij} = f_{ij} \tag{1}$$

A more common weighting approach is the so-called *tf•idf* (term frequency - inverse document frequency) weighting [17], which takes into account the frequency of the word throughout all documents in the collection along with the fact that documents may be of different lengths:

$$a_{ij} = \frac{f_{ij}}{\sqrt{\sum_{l=1}^{M} f_{lj}^2}} \times log\left(\frac{N}{n_i}\right) \tag{2}$$

For matrix **A**, the number of rows corresponds to the number of words M in the document collection. There could be hundreds of thousands of different words. In order to reduce the high dimensionality, stop-word (frequent word that carries no information) removal, word stemming (suffix removal) and additional dimensionality reduction techniques, feature selection or re-parameterization [12], are usually employed.

To classify a class-unknown document vector X, the *k*-Nearest Neighbor classifier algorithm ranks the document's neighbors among the training document vectors, and uses the class labels of the k most similar neighbors to predict the class of the new document. The classes of these neighbors are weighted using the similarity of each neighbor to X, where similarity is measuredby Euclidean distance or the cosine value between two document vectors. The cosine similarity is defined as follows:

$$sim(X, D_j) = \frac{\sum_{t_i \in (X \cap D_j)} x_i \times d_{ij}}{\|X\|_2 \times \|D_j\|_2} \tag{3}$$

where X is the test document; D_j is the jth training document; t_i is a word shared by X and D_j ; x_i is the weight of word t_i in X; d_{ij} is the weight of word t_i in document D_j ; $\|X\|_2 = \sqrt{x_1^2 + x_2^2 + x_3^2 + ...}$ is the norm of X, and $\|D_j\|_2$ is the norm of D_j. A cutoff threshold is needed to assign the new document to a known class.

The *k*NN classifier is based on the assumption that the classification of an instance is most similar to the classification of other instances that are nearby in the vector space. Compared

to other text categorization methods such as Bayesian classifier, *k*NN doesn't rely on prior probabilities, and it is computationally efficient. The main computation is the sorting of training documents in order to find the *k* nearest neighbors for the test document. More importantly, in a dynamic environment that requires frequent additions to the training document collection, incorporating newt raining documents is easy for the *k*NN classifier.

References

[1] S. Axelsson, "Intrusion Detection Systems: A Survey and Taxonomy", http://citeseer.nj.nec.com/axelsson00intrusion.html, 2000.

[2] V. N. P. Dao and V. R. Vemuri, "Computer Network Intrusion Detection: A Comparison of Neural Networks Methods", *Differential Equations and Dynamical Systems*, (Special Issue on Neural Networks, Part-2), vol.10, No. 1&2, 2002.

[3] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Logstaff, "A Sense of Self for Unix process", *Proceedings of 1996 IEEE Symposium on Computer Security and Privacy*, 120-128, 1996.

[4] S. Forrest, S. A. Hofmeyr and A. Somayaji, "Computer Immunology", *Communications of the ACM*, Vol. 40, 88-96, 1997.

[5] W. Lee, S. J. Stolfo and P. K. Chan, "Learning Patterns from Unix Process Execution Traces for Intrusion Detection", *Proceedings of AAAI97 Workshop on AI Methods in Fraud and Risk Management*, 50-56, 1997.

[6] C. Warrender, S. Forrest and B. Pearlmutter, "Detecting Intrusions Using System Calls: Alternative Data Models", *Proceedings of 1999 IEEE Symposium on Security and Privacy*, 133-145, 1999.

[7] A. K. Ghosh, A. Schwartzbard and A. M. Shatz, "Learning Program Behavior Profiles for Intrusion Detection", *Proceedings of 1st USENIX Workshop on Intrusion Detection and Network Monitoring*, Santa Clara, CA, April 1999.

[8] Y. Yang, "Expert Network: Effective and Efficient Learning from Human Decisions in Text Categorization and Retrieval", *Proceedings of 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'94)*, 13-22, 1994.

[9] C. Ko, G. Fink and K. Levitt, "Automated Detection of Vulnerabilities in Privileged Programs by Execution Monitoring", *Proceedings of 10th Annual Computer Security Applications Conference*, Orlando, FL, Dec, 134-144, 1994.

[10] M. Asaka, T. Onabuta, T. Inoue, S. Okazawa and S. Goto, "A New Intrusion Detection Method Based on Discriminant Analysis", *IEEE TRANS. INF. & SYST.*, Vol. E84-D, No. 5, 570-577, 2001.

[11] N. Ye, X. Li, Q. Chen S. M. Emran and M. Xu, "Probabilistic Techniques for Intrusion Detection Based on Computer Audit Data", *IEEE Trans. SMC-A*, Vol. 31, No. 4, 266-274, 2001.

[12] K. Aas and L. Eikvil, T*ext Categorisation: A Survey*, http://citeseer.nj.nec.com/aas99text.html, 1999.

[13] MIT Lincoln Laboratory, http://www.ll.mit.edu/IST/ideval/.

[14] Sun Microsystems, *SunShield Basic Security Module Guide*, 1995.

[15] R. Lippmann, D. Fried, I. Graf, J. Haines, K. Kendall, D. McClung, D. Webber, S. Webster, D. Wyschograd, R. Cunninghan and M. Zissan, "Evaluating Intrusion Detection Systems: the 1998 DARPA off-line Intrusion Detection Evaluation", *Proceedings of the DARPA Information Survivability Conference and Exposition*, IEEE Computer SocietyPress, Los Alamitos, CA, 12-26, 2000.

[16] K. Kendall, "A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems", Master's Thesis, Massachusetts Institute of Technology, 1998.

[17] J. T.-Y. Kwok, "Automatic Text Categorization Using Support Vector Machine", *Proceedings of International Conference on Neural Information Processing*, 347-351, 1998.

[18] M. Damashek, "Gauging Similarity with n-Grams: Language-Independent Categorization of Text", Science, Vol. 267, 843-848, Feb. 1995.