A Layered Detection Method for Malware Identification

Ting Liu, Xiaohong Guan, Yu Qu, Yanan Sun

SKLMS Lab and MOE KLNNIS Lab, Xi'an Jiaotong University, P. R. China

{tingliu, xhguan, yqu, ynsun}@sei.xjtu.edu.cn

Abstract. In recent years, millions of new malicious programs are produced by a mature industry of malware production. These programs have tremendous challenges on the signature-based anti-virus products and pose great threats on network and information security. Machine learning techniques are applicable for detecting unknown malicious programs without knowing their signatures. In this paper, a *Layered Detection* (*LD*) method is developed to detect malwares with a two-layer framework. The *Low-Level-Classifiers* (*LLC*) are employed to identify whether the programs perform any malicious functions according to the API-calls of the programs. The *Up-level-Classifier* (*ULC*) is applied to detect malwares according to the low level function identification. The *LD* method is compared with many classical classification algorithms with comprehensive test datasets containing 16135 malwares and 1800 benign programs. The experiments demonstrate that the *LD* method outperforms other algorithms in terms of detection accuracy.

Keywords: Machine learning; Network security; Malware detection; Malicious function identification

1 Introduction

Exponential growth and development of Internet has created unprecedented opportunities to access and share information. Millions of online users and ubiquitous Internet accesses have cultivated not only a great number of convenient services and fast-developing companies, but also innumerable malicious programs and a huge underground economy. Malicious tools enable attackers to gain access to a variety of valuable resources such as identities, credentials, hacked hosts, and other information and services. The attackers sell the information and services on the underground

market, and then reinvest the profits into the development of new malicious tools and services. This profitable cycle leads to a dramatic increase in the significant proliferation of malicious codes. In 2008, Symantec detected 1,656,227 new malicious programs which took up over 60% of the approximately 2.6 million malicious programs that Symantec has detected in total over time[1].

Malware is defined as a program that performs a malicious function, such as compromising a system's security, damaging a system or obtaining sensitive information without the user's permission. The malwares can be generally divided into the following categories: viruses, worms, Trojans and others. In recent years, Trojans and worms have been ranked as the top security threats, which occupied 68% and 29% of all new malwares and took over all top 10 malware families in 2008[1]. This paper concentrates on the Trojan and worm classification.

Due to inestimable damages caused by malicious programs, malware detection has become one of the primal research interests in the field of network and information security. Over the past decade, the signature-based detection, employed by most commercial antivirus products, such as Kaspersky Anti-Virus (KAV) and Symantec Norton AntiVirus, is the most widely used for detecting malware. In 2007, Kaspersky added 250,000 new signatures into their antivirus databases and released more than 10,000 database updates [2]. The rapidly-growing signature database would cause significant increase of computational cost to check all the signatures, and missing frequent updates would cause security loop holes.

Dynamic analysis is considered as an effective approach to detect unknown malwares. In 1991, researchers managed to determine what the programs attempt to do by observing and analyzing their actions at run-time in a controlled environment [3]. The major drawback of this technique is that the detection can only be done after at least a part of the malicious code is executed and has generated suspicious behaviors in the system. Therefore, many researchers have proposed that the suspected programs could be run and analyzed on the virtual machine (VM) before proceeding on an actual system [4, 5]. Since more and more malwares do not carry out malicious activities directly but only "activated" by a particular instruction, this method may not be effective in "normal" operation.

Applying machine learning in network security is regarded as one of the alternative effective solutions with integrated consideration of security and privacy with massive and ubiquitous information. The ability of machine learning to detect unknown attacks is demonstrated in the context of intrusion detection systems[6-8]. In recent

years, many machine learning techniques have been applied to detect unknown malwares. Machine learning approach is based on the assumption that malwares have certain characteristics not presented in benign programs. The supervised classifier is trained to distinguish malicious codes based on the known instances of malicious and benign programs [9]. Thereby, the performance of this method critically depends on the methods of feature extraction, feature selection and classification. Another issue is that few of machine learning based approach could identify the malware's functions which are useful for selecting defense strategy.

In this paper, a *Layered Detection (LD)* method is developed to detect malwares with a two-layer framework. The *Low-Level-Classifiers (LLC)* are employed to identify whether the programs perform any malicious functions according to the API-calls of the programs. The *Up-level-Classifier (ULC)* is applied to detect malwares according to the low level function identification. A hybrid structure (Type-function), constituting of the classification results of *LLC* and *ULC*, is proposed to describe the malware. The *LD* method is compared with many classical classification algorithms with comprehensive test datasets containing 16135 malwares and 1800 benign programs. The experiments demonstrate that the *LD* method outperforms other algorithms in terms of detection accuracy. Moreover, the Type-function is proved as an unprejudiced and effective method for describing malware functions. In fact, the *LD* method can accurately identify 98% of all malwares' functions, much higher than the other methods do. Furthermore, we also apply our method to distinguish malicious and benign programs. The experimental results demonstrated that the *LD* method integrated with *J48 Tree* outperformed others classifiers including *Boosting-J*48.

The rest of this paper is organized as follows. Section 2 presents the methods of feature extraction, feature selection, classification and measurement which will be used in this paper. The framework of the *LD* method is introduced in the Section 3. The experimental results are discussed in Section 4, which show the *LD* method outperforms other methods. Section 5 is the conclusions of this paper.

2 Machine Learning Based Approach

The primary goal of this study is to explore the performance of various machine learning techniques in detecting known and unknown malwares based on the API calls. A large number of programs, including the benign and the malicious, have been collected to train and evaluate the classifiers. The methods of feature extraction and

definition are firstly presented in this section. And then, the feature selection methods, classification algorithms and performance measurements are briefly described.

2.1 Feature Extraction and Definition

17935 Windows PE programs have been gathered, consisting of 16135 malicious and 1800 benign executables. The malicious executables are collected from malwares database of Kaspersky Corporation and Honey-Net in our lab, including 6377 Downloader Trojans, 4795 Spy Trojans, 2214 PSW Trojans, 1458 Email Worms, 532 P2P Worms and 759 Net Worms (which are expressed as M_1 to M_6 respectively). The benign programs are collected from a freshly installed Windows XP SP2 system, including DLL and EXE files. 2710 standard Windows API calls [10] are selected as the attributes of the program in this paper. By searching the Import Address Table of the programs, the Windows API calls of the programs are extracted and recorded in original dataset. The attribute is defined as $\{A_1, A_2, \ldots A_i, ..., A_{2710}, R\}$, where A_i identifies the *ith* API call (absence is 0 and presence is 1) and R represents the program category (benign is 0 and malicious is 1).

To facilitate the presentation, the notations are defined as follow:

$\boldsymbol{R}$ is the type of the sample. R_B, R_M and R_{Mi} means the sample is a benign, malicious and *i*th malware type program.

$\boldsymbol{A_i}$ is the *i*th API, A_{i1} means that the *ith* API is called; and A_{i0} means not.

$\boldsymbol{S}$ is the set of samples and defined as $\{A_1, A_2, \ldots A_m, R\}$.

2.2 Performance Measures

To measure the performance of previous feature selection and classification algorithms, several performance measures are selected in this work to represent and compare the results.

Confusion Matrix constitutes of the statistics of actual and predicted values. True Positive (*TP*) and True Negative (*TN*) are the number of correctly classified malicious and benign programs. False positive (*FP*) and False negative (*FN*) are the number of falsely classified benign and malicious programs. $N(R_M)$ and $N(R_B)$ are the number of the actual malicious and benign programs. $N(R_{PM})$ and $N(R_{PB})$ are the number of the predicted malicious and benign programs. N is the number of all samples.

True Positive Rate (TPR) is the rate of correctly classified malwares, which presents the detection rate.

$$TPR = TP/(TP + FN) = TP/N(R_M)$$

False Positive Rate (FPR) is the rate of misclassified benign programs in benign class.

$$FPR = FP/(TN + FP) = FP/N(R_B)$$

Accuracy is the rate of the entire correctly classified instances in whole set.

$$Accuracy = (TP + TN)/N$$

Kappa is used to measure the agreement between predicted and observed categorizations of a dataset, while correcting for agreement that occurs by chance.

$$Kappa = (Accuracy - Pe)/(1 - Pe)$$

where *Pe* is the hypothetical probability of chance agreement, using the observed data to calculate the probabilities of each observer randomly selecting category[11].

2.3 Feature Selection

A large number of features in many domains result in a huge challenges on the efficiency and accuracy of the classification. Typically, some of the features do not contribute to the accuracy of the classification task and may even hamper it. Therefore, identifying the most representative features is significant in minimizing the classification error and the resource consumption. In present research, three measurements: *Information Gain, One-Rule* [12] and *Chi-Square* [13] are employed to evaluate the contribution of each API in malware classification. The top ranked features are selected.

2.4 Classification Algorithm

Naive Bayes is one of the most successful learning algorithms for text categorization. It is based on the Bayes rule assuming conditional independence between classes. In this study, Naive Bayes algorithm is implemented by *WEKA NaiveBayes(**NB**) classifier* [14].

Decision Tree is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. In decision tree classifiers, the internal nodes are tests on individual

features, and leaves are classification decisions. Typically, a greedy top-down search method is used to find a small decision tree that correctly classifies the training data [13]. *C*4.5, a classical algorithm for generating a decision tree and an extension of *ID3* algorithm, is performed by *WEKA* ***J48*** *classifier* in this study.

Boosting refers to a general and provably effective method of producing a very accurate prediction rule by combining rough and moderately inaccurate rules of thumb. This technique trains successive component classifiers with a subset of the training data that is "most informative" given the current set of component classifiers. Classification of a test point is based on the outputs of the component classifiers. Boosting can achieve a very low training error, even a vanishing training error if the problem is separable[15]. *Adaptive Boosting (AdaBoost)* is one of the most popular boosting algorithms, formulated by Yoav Freund and Robert Schapire. In this work, it is used in conjunction with NB and J48 to improve their performance, which is executed by WEKA **AdaBoost** classifier [14].

3 Layered Framework and Method for Detecting Malwares

Many machine learning techniques have been applied to detect unknown malwares. In our study, it is demonstrated that the common classification algorithms are significantly affected by: 1) the imbalanced numbers of benign and malicious programs; 2) the known differences among various malwares; 3) the multiple correlated functions of malwares.

For most classification algorithms, the imbalanced numbers of different classes results in un-uniform classification criteria of different classes. In fact, most users only use a number of required software products, but are threatened by thousands of malwares. In this paper, 16135 malwares and 1800 Windows XP SP2 initial executables are collected as the training and testing dataset. When the *NB* and *J*48 classifiers are applied to categorize all these programs with 10-fold cross-validation, the *FPR* of them are as high as 32% and 18%.

Furthermore, the differences among various malwares also cause many difficulties for malware detection. Table 2 displays the probabilities of 10 APIs to be called by various programs, which represents the highest *Information Gain* in all APIs. It is noticed that *ReadProcessMemory* is scarcely called by the benign programs, Downloader Trojan, Email Worm and P2P Worm, but called by 15.1% of PSW Trojans. Obviously, the calling of *ReadProcessMemory* is a positive evidence to identify a program as

the *PSW Trojan*. However, it is a dilemmatic evidence for the classifiers to identify whether that program is malicious when all malwares are set in one class.

Table 1. The probability of API to be called by various of programs

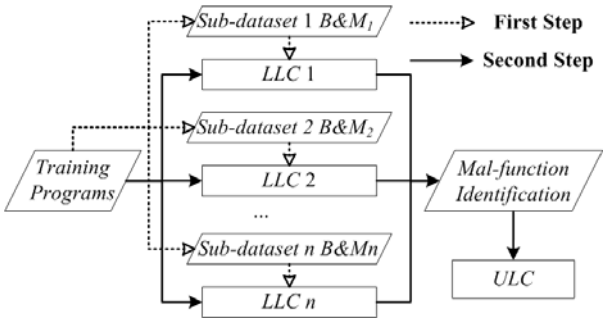
API	Probability to be called						
	Benign	**DLoader**	**PSW**	**SPY**	**Email**	**P2P**	**Net**
DisableThreadLibraryCalls	39.5%	2.6%	0.3%	1.9%	0.0%	0.0%	0.5%
InterlockedIncrement	53.8%	7.4%	3.4%	13.5%	7.4%	11.5%	15.0%
InterlockedDecrement	53.2%	9.0%	3.3%	13.4%	7.8%	11.1%	14.0%
GetLastError	77.2%	26.7%	18.4%	26.8%	22.2%	29.9%	30.3%
SetLastError	45.2%	6.1%	5.2%	6.0%	3.4%	2.6%	6.1%
ExitProcess	17.6%	62.5%	64.7%	63.8%	75.0%	79.3%	68.4%
HeapFree	40.1%	12.3%	4.7%	10.3%	11.1%	9.2%	14.9%
GetProcAddress	59.3%	78.6%	92.4%	88.6%	90.2%	89.8%	93.3%
ReadProcessMemory	***0.9%***	***1.5%***	***15.1%***	***4.2%***	***1.0%***	***0.8%***	***1.8%***
HeapAlloc	40.4%	15.0%	5.7%	10.4%	11.2%	9.2%	14.9%

The malware's type is useful and important for selecting defense strategy. However, it is extremely difficult to identify the type, since the most malwares perform multiple correlated functions. Even the security experts and companies define the malwares depending on their subjective judgments. For example, the *Rustock* is employed to create one of today's most extensive zombie networks for sending spam, exploiting the *Rootkit* techniques [16]. It is defined as *Backdoor* by Symantec, and as *Rootkit* by Kaspersky and MCafee.

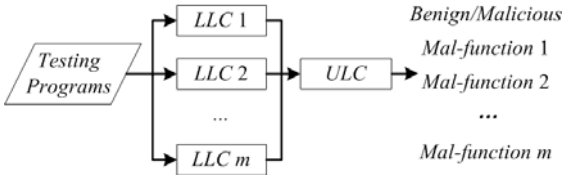
To address the above issues, a new method - *Layered Detection (LD)* is proposed for malware detection. It applies a two-layer framework to identify the malware's function and detect the malware, and uses a Type-function structure to express the malware classification result. The *LD* method will be introduced from two phases: training and testing.

As illustrated in Fig.1.a, the training phase of *LD* classifier is constituted of two steps. In the first step, all malwares are divided into several sub-datasets according to their type. These sub-datasets are applied to train the *Low-Level-Classifiers (LLC)* for identifying whether the programs perform the various malicious functions. Since these sub-datasets present better balance between the benign and malicious programs and retain the characteristics of various malware types, the *LLC* can identify the program's malicious functions with low *FPR* and high *Accuracy*. In the second step, the training programs are evaluated by all *LLCs* in parallel. The identification results are

used to train the *Up-level-Classifier (ULC)* for detecting malwares. If a *LD* classifier employs *NB* and *J*48 to train the *LLCs* and *ULC*, it is named as *LD-NB-J*48.



a. Training phase



b. Testing phase

Fig. 1. The model of Layered Detection Classifier

In the testing phase, all programs are first evaluated by the *LLC*s to identify whether they perform malicious functions, and are then estimated by the *ULC* to determine whether they are malwares, as shown in Fig.1.b. Both classifications of LLCs and ULC are output as Type-function structure for providing unprejudiced and comprehensive information to users.

4 Experiment and Discussion

4.1 Feature Selection

In all 2710 Windows standard APIs, there are 1559 APIs that are never called by any program. These APIs seldom contribute to the malware classification, but reduce the accuracy and exponentially increased the computation consumption. By filtering these useless APIs, a new subset was generated, consisting of the rest 1151 APIs (called as S_{1151}), to replace the original dataset in the following part. Then, three supervised feature selection methods: *IG*, *One-Rule* and *Chi-Square* are employed to evaluate

and rank the value of all APIs on classification. For each selection method, the top 25, 50, 75, 100, 150, 200, 250, 300, 400, 600, 800 and 1000 valuable attributes are selected as new sets. (These sets were marked as $S_{Method\ Num}$, for example S_{IG100} expresses 100 APIs are selected with *IG* measure.) The *NB* and *J*48 classifiers are applied to categorize all subsets using 10-fold cross-validation. For reliable results, every classifier is repeated 10 times on each set. The average *Kappa* is calculated to evaluate various feature selection methods, feature sizes, and classification algorithms.

Kappa describes the agreement between predicted and actual value by deducting the random success from the classifier's success. Hence, it is employed to evaluate the methods of feature selection and classification which showed oppositely on accuracy and *FPR* in this study.

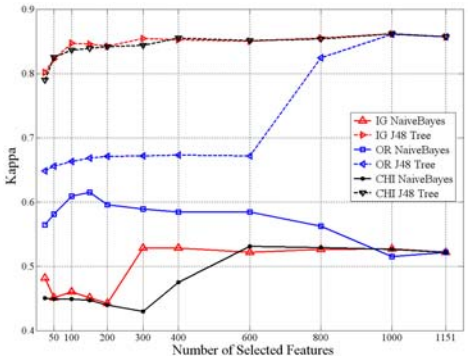


Fig. 2. Kappa of 3 feature selection methods

As illustrated in Fig 2, the *Kappa* of *NB* classifier is 0.522 on the S_{1151}. On the *OR* sets, it increases slowly but surely when more and more features are removed and exceeds 0.6 when the feature size is 100 and 150. On the *IG* and *CHI* sets, it quickly reduces about 20% when the feature size decreases from 600 to 200. That proves the *OR* method selects more effective and reliable features for the *NB* classifier than the *IG* and *CHI*. Considering the computational costs of classification, the feature size is expected to be as small as possible. As demonstrated in Fig.2, *J*48 classifier presents integrated performance on the S_{IG100}. The *Kappa* is more than 0.6 only on the S_{OR100} and S_{OR150} (the *Kappa* of a substantial classifier should be more than 0.6 [11]). Therefore, the IG and OR are chosen for the *J*48 and *NB* classifier, and the feature size is 100.

4.2 Malware Classification

In this subsection, we employ many classification algorithms to detect malwares, and compare their performance using different measurements. Besides *J*48 and *NB*, the *LD* method employs *Multilayer Perceptron* (*MP*) and *IBk* as its *ULC* algorithm. *MP* is a back propagation neural network classifier, and *IBk* is a k-nearest-neighbor classifier that uses the Euclidean distance metric [14, 15]. (*k* is set as 6 in this paper.) Moreover, the *AdaBoost*, collaborating with *J*48 and *NB* algorithms, is employed to compare with the *LD* method.

Table 2. Malware detection collaborating with *LD-J*48 on S_{IG100}

Method	***TP***	***TN***	***FP***	***FN***	***FPR***	***Accuracy***	***Kappa***
*J*48	15986	1484	316	149	17.56%	97.41%	0.8503
*Multi-J*48	15935	1523	277	200	15.39%	97.34%	0.8499
*AdaBoost-J*48	16016	1590	210	119	11.67%	98.17%	0.8961
*LD-J*48-*J*48	16036	1639	161	99	8.94%	98.55%	0.9185
*LD-J*48-*NB*	15369	1734	66	766	3.67%	95.36%	0.7809
*LD-J*48-*MP*	16040	1640	160	95	8.89%	98.58%	0.9200
*LD-J*48-*IBk*	16035	1636	164	100	9.11%	98.53%	0.9172

Table 3. Malware detection collaborating with *LD-NB* on S_{OR100}

Method	***TP***	***TN***	***FP***	***FN***	***FPR***	***Accuracy***	***Kappa***
NB	15860	982	818	275	45.44%	93.91%	0.6103
Multi-NB	15101	1247	553	1034	30.72%	91.15%	0.5620
AdaBoost-NB	15872	976	824	263	45.78%	93.94%	0.6104
LD-NB-NB	15277	1313	487	858	27.06%	92.50%	0.6195
*LD-NB-J*48	15674	1218	582	461	32.33%	94.18%	0.6680
LD-NB-MP	15731	1147	653	404	36.28%	94.11%	0.6523
LD-NB-IBk	15667	1220	580	468	32.22%	94.16%	0.6672

Various classification algorithms are applied to detect malwares according to the program's function which are identified by the *LD-J*48. As reported in Table 3, *LD-J*48, *LD-J*48-*MP* and *LD-J*48-*IBk* misclassify no more than 100 malwares; and their *Accuracy* and *Kappa* are as high as 98.5% and 0.92. Although the *LD-J*48-*NB* shows the worst *Accuracy*, it falsely classifies only 95 benign programs (the *FRP* is as low as 3.67%). The *AdaBoost* method obviously improves the ability to identify malwares of *J*48: the *FP* and *FN* reduce 33.5% and 20.1%, and the *Kappa* increases from 0.8503 to 0.8961. The *Multi-J*48 falsely classifies fewer benign programs, but misclassifies more malwares than the *J*48.

Table 4 shows the malware detection results when various methods are collaborated with *LD-NB*. Generally speaking, the performances of *LD-NB* on dataset S_{OR100} are not as good as that of *LD-J*48 on S_{IG100}, and the performances of various methods collaborating with *LD-NB* are similarly to the *LD-J*48.

The experimental results demonstrate that the *LD* method significantly improves the detection accuracy of many classifiers. In more specific terms, *LD-J*48-*MP* is recommended to pursue high *Accuracy* and *Kappa*, and *LD-J*48-*NB* is suggested to chase low *FPR*.

5 Conclusions

In paper we demonstrate that the accuracy of malware detection is significantly affected by: 1) the imbalanced numbers of benign and malicious programs; 2) the known differences among various malwares; 3) the multiple correlated functions of malwares. A new method based on machine learning is developed for malware detection. To address the above issues, a layered detection framework is established and the malwares are divided into several *sub-datasets* to train the *low-level-classifiers* for identifying malicious functions and are detected based on the identified functions by the up-level-classifier. Since the malwares can be well categorized in the framework, it is helpful for selecting defense strategy.

The experiments with 17935 collected programs show that the new method can correctly identify the functions of 98.5% malwares and detect 98.6% of all malwares. Moreover, many feature selection methods and classification algorithms are also investigated. It is demonstrated that the *IG* and *OR* feature selection methods are amenable for *J*48 and *NB* classifiers respectively and the best feature size is 100. The new method demonstrates more than 95% accuracy, about twice as high as *Multi-J*48 and *Multi-NB*, for malicious function identification, and outperforms other classification algorithms including *Boosting* for malware detection.

Reference

[1] Gostev, A.: Kaspersky Security Bulletin: Statistics 2008.: (2009)

[2] Lo, R., Kerchen, P., Crawford, R., Ho, W., Crossley, J., Fink, G., Levitt, K., Olsson, R., Archer, M.: Towards a testbed for malicious code detection.: Compcon Spring '91. Digest of Papers (1991) 160-166

[3] Wang, X., Yu, W., Champion, A., Fu, X., Xuan, D.: Detecting worms via mining dy-

namic program execution.: Security and Privacy in Communications Networks and the Workshops, 2007. SecureComm 2007. Third International Conference on (2007) 412-421

[4] Jiang, X., Wang, X., Xu, D.: Stealthy malware detection and monitoring through VMM-based "out-of-the-box" semantic view reconstruction. ACM Transactions on Information and System Security 13 (2010)

[5] Wenke, L., Stolfo, S.J., Mok, K.W.: A data mining framework for building intrusion detection models.: Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on (1999) 120-132

[6] Berral, J.L., Poggi, N., Alonso, J., Gavald, R., Torres, J., Parashar, M.: Adaptive distributed mechanism against flooding network attacks based on machine learning.: Proceedings of the 1st ACM workshop on Workshop on AISec. ACM, Alexandria, Virginia, USA (2008) 43-50

[7] Kloft, M., Brefeld, U., D, P., essel, Gehl, C., Laskov, P.: Automatic feature selection for anomaly detection.: Proceedings of the 1st ACM workshop on Workshop on AISec. ACM, Alexandria, Virginia, USA (2008) 71-76

[8] Renchao, Q., Tao, L., Yu, Z.: An immune inspired model for obfuscated virus detection.: 2009 International Conference on Industrial Mechatronics and Automation, ICIMA 2009, Chengdu, China (2009) 228-231

[9] Windows: Windows API Reference.: http://msdn.microsoft.com/en-us/library/aa383749(VS.85).aspx

[10] Landis, J.R., Koch, G.G.: The measurement of observer agreement for categorical data. Biometrics 33 (1977)

[11] Holte, R.C.: Very simple classification rules perform well on most commonly used datasets. Mach Learn 11 (1993) 63-91

[12] Moskovitch, R., Elovici, Y., Rokach, L.: Detection of unknown computer worms based on behavioral classification of the host. Computational Statistics and Data Analysis 52 (2008) 4544-4566

[13] Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques (second ed.). Morgan Kaufmann, San Francisco (2005)

[14] Duda, R.O., Hart, P.E., Stork, D.G.: Pattern Classification (2nd Edition). Wiley-Interscience (2000)

[15] Gostev, A.: Rustock and All That.: (2008)