Classification of Malicious Web Code by Machine Learning

Ryohei Komiya
University of Aizu,
Ikkimachitsuruga,
Aizuwakamatsu City,
Fukushima, 965-8580 Japan
rkomiya.pub@gmail.com

Incheon Paik
University of Aizu,
Ikkimachitsuruga,
Aizuwakamatsu City,
Fukushima, 965-8580 Japan
paikic@u-aizu.ac.jp

Masayuki Hisada
NSTLAB.,
Ikkimachitsuruga,
Aizuwakamatsu City,
Fukushima, 965-0006 Japan
info@nstlab.com

***Abstract*—Web applications make life more convenient through on the activities. Many web applications have several kind of user input (e.g. personal information, a user's comment of commercial goods, etc.) for the activities. However, there are various vulnerabilities in input functions of web applications. It is possible to try malicious actions using free accessibility of the web applications. The attacks by exploitation of these input vulnerabilities enable to be performed by injecting malicious web code; it enables one to perform various illegal actions, such as SQL Injection Attacks (SQLIAs) and Cross Site Scripting (XSS). These actions come down to theft, replacing personal information, or phishing. Many solutions have devised for the malicious web code, such as AMNESIA [1] and SQL Check [2], etc. The methods use parser for the code, and limited to fixed and very small patterns, and are difficult to adapt to variations. Machine learning method can give leverage to cover far broader range of malicious web code and is easy to adapt to variations and changes. Therefore, we suggests adaptable classification of malicious web code by machine learning approach such as Support Vector Machine (SVM)[3], Naïve-Bayes[4], and k-Nearest Neighbor Algorithm[5] for detecting the exploitation user inputs.**

Keywords-component; Security, Web Application, Machine Learning

I. INTRODUCTION

Currently, many web applications have security problems in the application layer. The vulnerabilities in the security problems are exploited by crackers. Crackers typically attack web applications by injecting or changing a query of HTTP request or response.

They usually pay attention to the two types of malicious code; SQLIAs and XSS [6]. The purpose of SQLIAs is to control databases in a web application illegally. SQLIAs enable to steal or change important data (e.g. addresses, and credit card number, etc.) in databases. The purpose of XSS is to inject malicious scripts in a web application. When users access the web application which is injected XSS, injected scripts do bad things on the user's web browser (e.g. phishing, and stealing cookies, etc.).

Existing approaches can detect registered patterns like existing malicious web code with high accuracy. There are two types, positive security model and negative security model, in existing approaches for detecting malicious web code. The positive security model registers many patterns of non-malicious web code, and it detects a query as a malicious web code when the query doesn't match up with registered patterns. In contrast, the negative security model registers many patterns of malicious web code, and it detects a query as a malicious web code when the query matches up with the registered patterns.

Currently, many solutions have been devised for detection of malicious code. The methods parse user input by parser, and confirm match limited to fixed and very small patterns which are modeled by reference to existing malicious web code. However, there are new malicious web code which can be created over the registered patterns in existing approaches deliberately. Therefore these are difficult to adapt to changed environment. Machine Learning method can overcome this problem because it can give leverage far broader range of malicious web code and is easy to adapt to variations and changes.

In this research, we create a novel classifier of malicious web code for evaluation of the approach for code classification. There are several types of machine learning algorithms, so we should investigate the aptitude of each machine learning algorithms. Therefore, each classifier uses different machine learning algorithms because of finding the appropriate method for classification of malicious code.

In this paper, we explain a background of malicious web code about SQLIAs and XSS in Section 2. We introduce researches which are related to our approach in Section 3. The detail of our approach is explained in Section 4. The evaluation result is explained in Section 5.

II. BACKGROUND OF MALICIOUS WEB CODE

A. SQL Injection Attacks

Many web applications use Relational Data Base (RDB) which is a popular method for management of many data. RDBs are generally controlled by statements written in Structured Query Language (SQL). When web applications are required any data from user, it requests the data to RDB by input SQL that is associated hardcode and user input. Generally, web applications suppose that user input is constructed by only just text data (e.g. queries for search engines, password, comments on articles, etc.). However, it also enable to input malicious SQL statements for rewriting SQL statements against the intention of managers and developers of a web application. These attacks are called SQLIAs. There are several types of SQLIAs.

First example of SQLIAs is an attack for unauthorized login by conversion of the WHERE statement as TRUE. The WHERE statement is a part of the SQL statements which is used to describe conditions for extracting data as follows:

SELECT * FROM *user_table* WHERE *id*='*user_id*' AND *password*='*pass*'; (1)

This SQL statement (1) requests to extract data from *user_table* whose *id* and *password* agree with *user_id* and *pass* for authorizing login. If a condition of the WHERE statement is always TRUE, it extracts all data in the table. Therefore, it becomes available to access all data in a table through the use of malicious SQL statements that can rewrite a result of the WHERE statements as follows:

' OR 1=1; -- (2)

For example, this malicious SQL statement (2) is injected to the SQL statement (1) as follows:

SELECT * FROM *user_table* WHERE *id*='' OR 1=1; -- ' AND *password*='*pass*'; (3)

The WHERE statement in this injected SQL statement (3) is always TRUE, because 1=1 is always TRUE. Therefore, this injected SQL statement (3) extracts all data from *user_table* despite *id* and *password* are not correct, so it becomes available unauthorized login. Several unnecessary symbols (' AND *password*='*pass*';) are ignored by line comment. Words after the symbol of line comment (--) are treated as comments.

Another example of SQLIAs is the attack for stealing data illegally by the UNION statement. The UNION statement is a part of SQL statements which is used to combine results of two SELECT statements as follows:

SELECT * FROM *products_table* WHERE *name*='*product_name*'; (4)

This SQL statement (4) requests to extract data from *products_table* whose *name* agree with *product_name* for searching products. In this case, SQLIAs come into effect by injecting the following code:

' UNION SELECT * FROM *user_table*; -- (5)

This malicious SQL statement (5) is injected to the SQL statement (4) as follows:

SELECT * FROM *products_table* WHERE *name*=' ' UNION SELECT * FROM *user_table*; -- '; (6)

This SQL query (6) is combined results of two SELECT statements; "SELECT * FROM *products_table* WHERE *name*=' '" and "SELECT * FROM *user_table*;" by the UNION statement. If web applications response search results by using the SQL statement like (6), it becomes available to display data in *user_table* with the search results by the injecting malicious SQL statement (5).

B. Cross Site Scripting

Many web applications have any dynamic user interface (e.g. input form, and image view, etc.). One of major script language for implementation of these user interfaces is JavaScript. JavaScript is normally written or imported in HTML pages. When users open web pages with JavaScript, it runs scripts on the web browser of the users. In many web pages, there are several points which are written any user input (e.g. bulletin board system, and user comments in weblogs, etc.). User inputs in these points are written by plane text format in HTML pages. Therefore, it enables to let web browsers of users run malicious scripts by injecting JavaScript in these points as a user input. These attacks are called XSS. There are several types of XSS.

First example of XSS is an attack for phishing. When JavaScript has the function which let users move the other web page. With this function, it becomes available to let a user open a phishing web site when the user open the web pages which are injected XSS by a cracker. These attacks come into effect by an injecting the following malicious script:

document.location="http://cracker.com/phishing.html"; (7)

This malicious script (7) let a user move the other web page "http://cracker.com/phishing.html". The cracker can prompt the user to input important personal information such as address, and credit card number by uploading an impersonation web page.

Another example of XSS is an attack for stealing cookies. JavaScript has the function for getting cookies. With this function, it becomes available to let a user transfer his/her cookies to the cracker when the user open the web pages which are injected XSS like the previous example. These attack come into effect by injecting the following malicious script:

document.location="http://cracker.com/cookie.cgi?cookie=" +document.cookie; (8)

This malicious script (8) let a user move the other web page "http://cracker.com/cookie.cgi" by the same way of the previous example with the parameter "cookie" mandatorily. The value of the parameter "cookie" is a result of the script "document.cookie" which is the function to get cookies. Therefore, if web page "http://cracker.com/cookie.cgi" is an application for storing a parameter, it becomes available to steal cookies of a user when the user opens the web page which are injected the malicious script (8).

III. RELATED WORK

The most popular method for detecting malicious web code is Pattern Match. This method classifies malicious web code when user input matches up with registered patterns. AMNESIA is a model-based technique like Pattern Match

for classification of SQLIAs. AMNESIA analyzes a web application statically for extracting several positions which sends SQL queries to Databases and generates models of normal SQL query. AMNESIA detects queries as SQLIAs if sent SQL queries don't match up with the generated models. Dynamic Tainting [7] is also similar Pattern Match. Dynamic Tainting treats all user input or processed data of user input as tainted data. When tainted data is used for unsafe processes (e.g. sending SQL queries, file access), Dynamic Tainting detects tainted data as malicious web code if tainted data don't match up with the registered patterns.

Malicious web code are also detected by Parsing Approach. This method parses user input along grammar. For example, SQL queries are parsed along the grammar of SQL. SQL Check is one of Parsing Approach. SQL Check analyzes syntaxes of SQL queries by the SQL parser then each user input is taken as an argument query. If argument queries don't match up with registered data type, SQL Check detects it as malicious web code. SQL Guard [8] is also one of Parsing Approach which is similar to SQL Check. SQL Guard extracts some positions which send SQL queries statically like AMNESIA, and it parses normal SQL queries of each point. When SQL queries are sent, SQL Guard it dynamically exchange. If a dynamically analyzed syntax doesn't match corresponding statically analyzed syntax, SQL Guard detects it as malicious code.

Others, there is the method which uses random numerals or strings of characters for detection. Random numerals or strings of characters are used to mark keywords written in hardcode. SQL rand [9] is included in this method. When SQL queries are sent, SQL rand append random numerals to SQL standard keyword written in hardcode dynamically. Then, the SQL standard keywords in malicious input are not appended it. When SQL queries which are appended random numerals are reached to a database server, SQL rand removes its random numerals and execute queries. If there are SQL standard keywords which are not appended random numerals in SQL queries, SQL rand detects it as malicious web code.

IV. PROPOSED SOLUTION

In this section, we propose a solution for detecting malicious web code.

Our novel approach enables to detect new malicious web code flexibly by utilizing machine learning algorithms. Machine learning algorithms realize learning function which is close to human beings. New malicious web code set is created over patterns of existing malicious web code, so it is difficult to define a border between malicious web code and non-malicious web code. However, machine learning algorithms can classify new malicious web code either the malicious web code class or the non-malicious web code class with high accuracy. Because machine learning algorithms learn some features of malicious web code from training datasets which are included many malicious web code and non-malicious web code and it derives a flexible criterion between malicious web code and non-malicious web code. Also, an accuracy of machine learning algorithms is affected by what it defines features. Generally, machine learning algorithms for classification of texts use terms separated by blanks as features. However, our approach realizes higher accuracy than existing machine learning approaches by extracting some tokens which consist of specific terms of malicious web code as features. In this research, we create classifiers for classification of malicious web code by using machine learning algorithms.

A. Classifier

Our classifiers classify an user input character string as either malicious code or non-malicious code. In our approach, we require two processes for classifying input; learning process and classification process. Figure 1 illustrates the flow of the processes. In the learning process, it must determine criterion for classification of input. The classifier extracts features of malicious web code or non-malicious web code from each training data as feature vector, and determines a criterion from feature vectors and its label by the machine learning algorithm. Labels describe the class belonging to training data after tokenizing it. In the classification process, the classifier classifies user input based on the criterion for classification that is determined in the learning process by the machine learning algorithm.

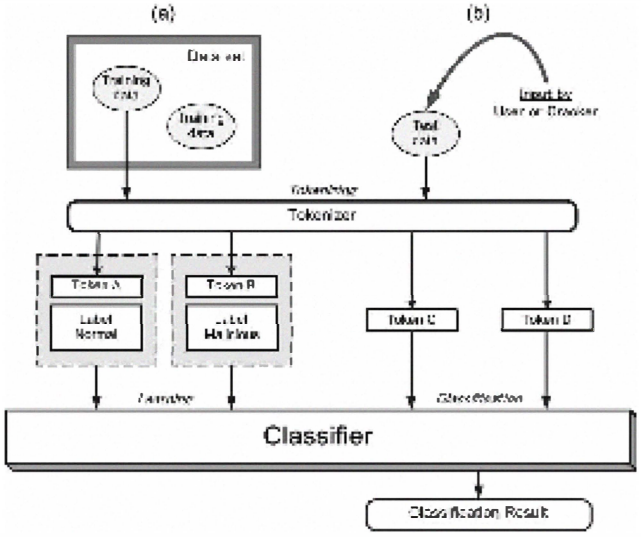

Figure 1. The flows of two classification processes: the learning process (a) and the classification process (b)

B. Generation of Feature Vector

A feature vector is abstraction format of features of data as documents which include both of training data and input. Many machine learning algorithms for classification to several classes need to extract features from documents as a format for learning and classification. A feature vector describes as follows:

$$(w_1, w_2, w_3, \ldots). \quad (4)$$

where w_x is a weight of each term. Each term is extracted from documents by any extraction method, and weights of

each term are calculated by any evaluation method such as TF-IDF method[10]. In this paper, we propose two extraction methods that Blank Separation Method and Tokenizing Method.

1) Blank Separation Method: Documents generally consist of combination of many terms which are separated by blank space. Blank Separation Method extracts terms with respect to each blank space. Number of each term is used for calculation of weight. For example, when you separate the following document:

‘ OR 1=1; -- (5)

Blank Separation Method separates this document as follows:

TABLE I. EXAMPLE OF BLANK SEPARATION METHOD

Name	Number of Term
‘	1
OR	1
1	2
=	1
;	1
--	1

2) Tokenizing Method: Malicious web code have some tokens which describes the feature of malicous web code. Tokenizing Method extracts terms with respect to each token which is defined based on our repetitive evaluation to determine better token as follows:

TABLE II. TOKENS OF SQLIAS

Name	Corresponding Terms
Line Comment	-- #
Comment	/* */
Operator	<> <=> >= <= == = != << >> < > | & - + % ^ ?
Logical Operator	NOT AND OR XOR ! && ||
Punctuation	[ ] ( ) , ; . ‘ “
Literal	“*any string*” ‘*any string*’ `*any string*`
Table Handling	SELECT INSERT UPDATE DELETE CREATE DROP ALTER RENAME
Numeral	*Numeral included in* +, -, e, *and* R
Reserve	*Other reserved words*
Ident	*Not reserved words consisted of alphabets and under bar ( ).*
Symbol	Other words

TABLE III. TOKENS OF XSS

Name	Corresponding Terms
Tag	*<any string> </any string>*
Punctuation	{ } ( ) [ ] . ; , <> <= >= == != === !== + - * % ++ -- << >> >>> & | ^ ! ~ && || ? : = + -= *= %= <<= >>= >>>= &= |= ^= / /=
Literal	“*any string*” ‘*any string*’
Numeral	*Numeral included in* +, -, e, *and* R
Object	*Object names of JavaScript*
FPM	*Functions, Methods, and Property of JavaScript*
Reserve	*Other reserved words*
Ident	*Not reserved words consisted of alphabets and under bar ( ).*
‘pass’	Other words

Numbers of each term is also used for calculation of weights. For example, when you separate the document (5), Tokenizing Method separates it as follows:

TABLE IV. EXAMPLE OF TOKENIZING METHOD

Name	Number of Term
Punctuatation	2
Logical Operator	1
Numerical	2
Operator	1
Line Comment	1

V. EVALUATION

We implemented and evaluated two classifiers both SQLIAs and XSS which can use TF-IDF method for weight calculation, and three machine learning approach among SVM, Naïve-Bayes, k-Nearest Neighbor Algorithm. Additionally, we evaluated some types of kernel functions in SVM, Linear Kernel[11], Polynomial Kernel[11], and Gaussian Kernel[11]. Also, we implemented the evaluation application for evaluation of two classifiers. In the learning process, this evaluation application reads training dataset from text files and puts each data to the learning methods of each classifier. The classifiers generate feature vectors from received data by TF-IDF method and learn it by each machine learning methods. In the classification process, this evaluation application also reads the test dataset from text files and puts each data to the classification methods of each classifier. The classifiers generate feature vector from received data and classify it. The classification results of each machine learning approach are analyzed by Precision, and Recall[12] in the evaluation application. These classifiers are developed by Java Platform, Standard Edition Development Kit 6[13].

We used several dataset for evaluation as follows:

TABLE V. DATASETS FOR SQLIAs EVALUATION

	Training Data	Test Data
Normal Code	347	137
Malicious Code	348	221

TABLE VI. DATASETS FOR XSS EVALUATION

	Training Data	Test Data
Normal Code	87	107
Malicious Code	89	180

These dataset are collected with the cooperation of experts in attacks to web applications.

Figures from figure 2 to figure 7 are evaluation results described by Recall-Precision graphs. Also, Tables, table 11 and table 12, are evaluation results described by Precision and Recall. The maximum accuracy of SVM is the highest for SQL Injection and XSS. Additionally, both precision and recall are high. Naïve-Bayes and k-Nearest Neighbor Algorithm are lower than SVM. The maximum accuracy of Gaussian Kernel is the highest in three kernel functions. These results lead that the classifier by SVM using Gaussian Kernel is the most appropriate classifier in our evaluation.

TABLE VII. EVALUATION RESULTS OF SQLIAs

	Accuracy	Precision	Recall
SVM (Linear Kernel)	98.60%	0.977	1.00
SVM (Polynomical Kernel)	98.60%	0.977	1.00
SVM (Gaussian Kernel)	99.16%	0.986	1.00
Naïve-Bayes	98.88%	0.986	0.995
k-Nearest Neighbor Algorithm	98.60%	0.991	0.986

TABLE VIII. EVALUATION RESULTS OF XSS

	Accuracy	Precision	Recall
SVM (Linear Kernel)	98.26	0.989	0.983
SVM (Polynomical Kernel)	98.61%	0.989	0.989
SVM (Gaussian Kernel)	98.95%	0.989	0.994
Naïve-Bayes	93.73%	1.000	0.909
k-Nearest Neighbor Algorithm	98.61%	0.983	0.994

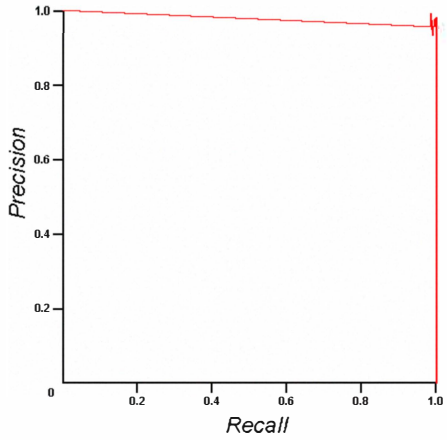

Figure 2. The Recall-Precision graph of SVM with Gaussian Kernel in SQLIAs.

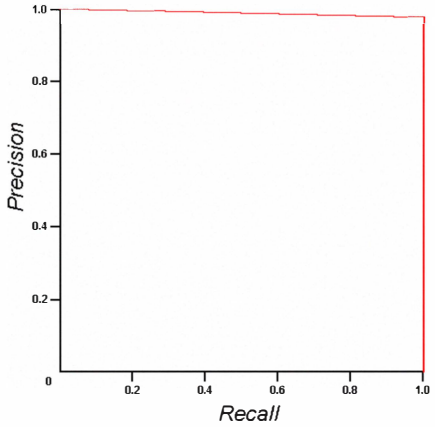

Figure 3. The Recall-Precision graph of SVM with Polynomical Kernel in SQLIAs.

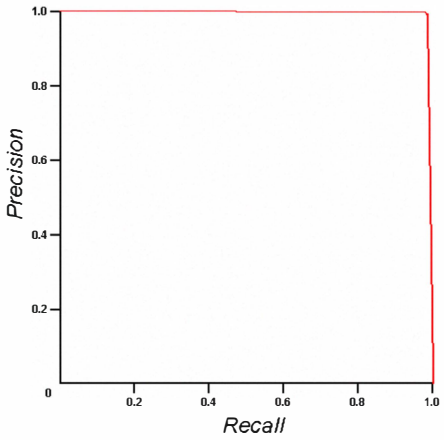

Figure 4. The Recall-Precision graph of k-Nearest Neighbor Algorithm in SQLIAs.

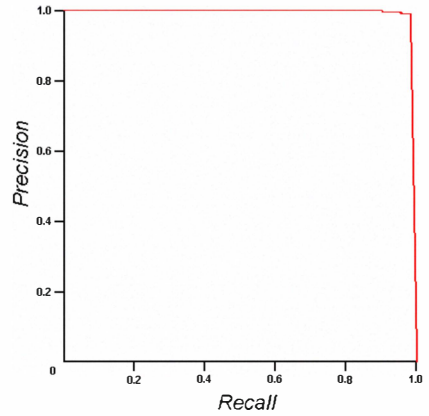

Figure 5. The Recall-Precision graph of SVM with Gaussian Kernel in SQLIAs.

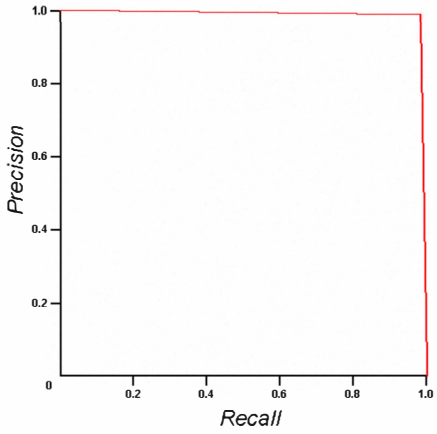

Figure 6. The Recall-Precision graph of SVM with Polynomical Kernel in SQLIAs.

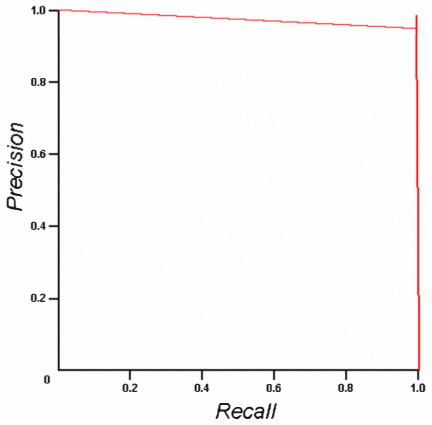

Figure 7. The Recall-Precision graph of k-Nearest Neighbor Algorithm in SQLIAs.

VI. CONCLUSION AND FUTURE WORK

Many web applications have any vulnerabilities such as SQLIAs and XSS. The vulnerabilities cause theft or defacing of important data by malicious web code. Existing provision cannot deal with new malicious web code flexibly. Our approach solves the problem using ability of characterizing and training by machine learning. Evaluation of our approach indicates high precision (SQLIA is 99.16% and XSS is 98.95%) by using SVM with Gaussian Kernel.

There are many types of attacks with malicious web code such as Directory Traversal, Error Handling, etc. We think that our solution may be efficiently corresponding to these malicious web code. We consider that the investigation for application of the solution to many types of malicious web code should be tried for the future code.

REFERENCES

[1] W.G.Halfond and A.Orso, "AMNESIA: Analysis and Monitoring for NEutralizing SQL-Injection Attacks," Proc. IEEE and ACM International Conference on Automatic Software Engineering (ASE 2005), Long Beach, CA, USA, Nov 2005.

[2] Z.Su and G.Wassermann, "The Essence of Command Injection Attacks in Web Applications," The 33rd Annual Symposium on Principles of Programming Language (POPL 2006), Jan 2006.

[3] C.Cortes and V.Vapnik, "Support vector networks," Machine Learning, 20, pp. 273-297, 1995.

[4] D.Pedro and M.Pazzani, "On the optimality of the simple Bayesian classifier under zero-one loss," Machine Learning, 29, pp. 103-137, 1997.

[5] Belur V.Dasarathy, "Nearest neighbor (NN) norms: Nn pattern classification techniques," IEEE Computer Society Press, 1991.

[6] The Open Web Application Security Project (OWASP). The Ten Most Critical Web Application Security Risks 2010. https://www.owasp.org/index.php/Top_10_2010-Main

[7] V.Haldar, D.Chandra, and M.Franz, "Dynamic Taint Propagation for Java," Proc. 21st Annual Computer Security Applications Conference, Dec 2005.

[8] G.T.Buehrer, B.W.Weide, and P.A.G.Sivilotti, "Using Parse Tree Validation to Prevent SQL Injection Attacks," International Workshop on Software Engineering and Middleware (SEM), 2005.

[9] S.W.Boyd and A.D.Keromytis, "SQLrand: Preventing SQL Injection Attacks," Proc. the 2nd Applied Cryptography and Network Security (ACNS) Conference, pp. 292-302, Jun 2004.

[10] K.Sparck Jones, "Statistical interpretation of term specificity and its application in retrieval," Journal of Documentation, 28(1), pp. 11-20, 1972.

[11] N.Cristianini and J.Shawe-Taylor, "An Introduction to Support Vector Machine and Other Kernel-based Learning Methods," Cambridge University Press, 2000.

[12] David L.Olson and D.Delen, "Advanced Data Mining Techniques," Springer; 1 edition, pp. 138, Feb 2008.

[13] ORACLE. Java Platform, Standard Edition 6 Release. http://www.oracle.com/technetwork/java/javase/overview/index-jsp-136246.html.