2012

Ensemble Methods for Malware Diagnosis Based on One-class SVMs

Xing An
Louisiana State University and Agricultural and Mechanical College

ENSEMBLE METHODS FOR MALWARE DIAGNOSIS BASED ON ONE-CLASS SVMS

A Thesis

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Master of Science

in

The Department of Computer Science

by
Xing An
B.E. Wuhan University 2008
May 2013

Acknowledgements

Sincere thanks to my major professor Dr. Jian Zhang, for his patient guidance throughout the process leading up to the completion of the thesis. I would also like to thank my committee members Dr. Jianhua Chen and Dr. Seung-Jong Park for their guidance and support. In addition, I want to extend my gratitude to all those in the Department of Computer Science for the help I acquired in my study and research.

I want to thank Mr. Hung-Te Lin for providing a tutorial in my mother tongue that helped me a lot in my initial study of LIBSVM and Mr. Zheng Lu for doing the proofreading of the thesis.

I wish to also thank my parents and grandparents for their eternal love, encouragement and support.

Table of Contents

Abstract

Malware diagnosis is one of today's most popular topics of machine learning. Instead of simply applying all the classical classification algorithms to the problem and claim the highest accuracy as the result of prediction, which is the typical approach adopted by studies of this kind, we stick to the Support Vector Machine (SVM) classifier and based on our observation of some principles of learning, characteristics of statistics and the behavior of SVM, we employed a number of the potential preprocessing or ensemble methods including rescaling, bagging and clustering that may enhance the performance to the classical algorithm.

We implemented the idea of rescaling by iteratively magnifying the attributes used by the support vectors of SVM and eliminating those unused ones from the training data examples until a maximum accuracy is achieved. Our study of bagging and clustering focused on the situation where only examples of malware are available and one-class SVM is used. For both methods, a group of models is built using part of the training data instead of building one model with the whole training data set. We also compared the effect of two possible coordination approaches for the sub-models acquired in the training process, namely, voting and one positive to be positive. Results of experiments showed that when utilized together with appropriate coordination methods, ensemble methods can effectively decrease both the cases where malware is labeled as clean or clean software is classified as malware, which are formally known as false-negative and false-positive errors in our context respectively.

Chapter1
Introduction

Malware, which is the abbreviation of malicious software, is a class of malicious code that includes worms, computer viruses, Trojan horse, etc. The purpose and effect of malware is to disrupt the normal functionality of computer system, gather and modify essential information, or to acquire illegal control of certain computer system [1]. Nowadays, with the development of the computer theory and technology, malware is no longer necessarily be a whole software; it could be a small section of code, scripts, or active content embedded in its carrier software, document or email. On the other hand, some of the malware can reproduce themselves, mark themselves as system files and even do self-propagation via the Internet, making it even more difficult to detect and identify them. As a result of this, static analysis of code is no longer enough to tell the malware from clean software. One of the most popular topics in the field of machine learning is classification which could be applied to the diagnosis of software if we consider malware as positive example and clean software as the negative example. Although we can direct apply some of the existed classification algorithms to this problem and some of them do perform already very well, we still like to explore the effects of some of the training techniques on this topic, which is the focus of this thesis.

Rescaling is based on the fact that only a proportion of the attributes provided in the training data are useful for classifiers while other attributes may even act as noise in this process [2]. Hence, one possible enhancement is to selectively allow part of the attributes to be used and eliminate those useless ones. One question of this approach is how to find and choose the useful attributes. In our implementation, this task is performed by parsing the SVM (Support Vector Machine) model stored in the form of plain text file; calculate the normalized weight according to each support vector; rescale the training data with these weights. The process of rescaling is

iterative and it will not stop until a maximum accuracy of prediction is reached. In our experiment, we use the n-gram approach to transform the encoded malware behavior into attributes for the SVM classifier, then we choose a group of data that are the most difficult to classify among the whole data set and compared the rescaling approach with the direct training approach on it [3]. According to the results generated in the experiment, we come to the conclusion that rescaling does effectively increase the accuracy of classification even if the data set is intrinsically complex.

Bagging [14] and clustering are two ensemble methods [4] we studied in the context of one-class SVM [5], that is, instead of being given the information of both the malware and clean software, we only have examples of malware. With bagging [14], we repeatedly sample a fixed proportion from the training data set, build a group of sub-models and make predictions with each one of them; for clustering, we conducted a bottom-up hierarchical clustering along with the greedy approach for selecting the candidates in each iteration, the clustering process terminates when all the clusters have merged into one final cluster that contains all the training examples. Apart from the two ensemble methods, we also studied two coordination approaches for reconciling the various predictions made by different models. Namely, they are voting [6] (final prediction is decided by the majority prediction of the sub-models) and one positive to be positive (as long as one of the sub-models claims an examples to be positive, the final prediction says positive). In our experiment, we studied the comparison of bagging, clustering and the traditional method under all the possible combinations of parameter settings. The result of our study implies that in the context of one-class SVM, ensemble methods with appropriate coordination do decrease both false-negative errors (malware is mistakenly labeled as clean) and false-positive errors [7]. (clean software is marked as malware).

This thesis is organized as follows: Chapter 2 explains the basic concepts and tools we used in our experiments such as SVM and n-gram, which is the foundation of further discussion of our work. Chapter 3 focuses on the rescaling methods. Implementation of the theory, settings and process of the experiment along with the results are shown. Chapter 4 describes our study of the ensemble methods, namely bagging and clustering, Chapter 5 summarizes our work and provides a brief outline for the future task to be carried out.

Chapter 2
Background

2.1 SVM

Support Vector Machines (SVMs) are supervised learning models with associated learning algorithms that analyze data and recognize patterns, used for classification and regression analysis [8]. An SVM model maps examples as points in a way such that examples of different classes are divided by a gap that is as wide as possible. Upcoming examples are then mapped into that same space and predicted as belonging to one of the categories according to which side of the gap they fall on.

Formally, the input for SVM is a training set D in the form [9]:

$$D = \{(X_i, y_i) | X_i \in R^p, y_i \in \{-1, 1\}\}_{i=1}^{n}$$

Here, y_i is the class label of the point X_i. And the goal of learning is:

Minimize: $||\mathbf{W}||$

subject to (for any i = 1, 2, 3, …, n):

$$y_i(W.X_i - b) \geq 1$$

As there is a chance that no hyperplane can separate the two classes thoroughly, we need to introduce a slack variable ξ to tolerate some mislabeled examples. By using the Lagrange multiplier, the problem is transformed to:

$$\min_{w,\xi,b} \max_{\alpha,\beta} \{\frac{1}{2}||W||^2 + C\sum_{i=1}^{n} \xi_i - \sum_{i=1}^{n} \alpha_i[y_i(w.X_i - b) - 1 + \xi_i] - \sum_{i=1}^{n} \beta_i \xi_i\}$$
$$\alpha_i, \beta_i \geq 0$$

The dual form is:

To Maximize:

$$L(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j k(X_i, X_j)$$

subject to (for any i = 1, 2, …, n)

$$0 \leq \alpha_i \leq C,$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

In addition to performing linear classification, SVMs can efficiently perform non-linear classification using kernel trick, implicitly mapping their inputs into high-dimensional feature spaces. For example, the Gaussian radial basis function (rbf) [10] is:

$$k(X_i, X_j) = e^{-\gamma ||X_i - X_j||^2}$$

Multi-class prediction is also enabled by using either one-to-one to one-to-multiple approach.

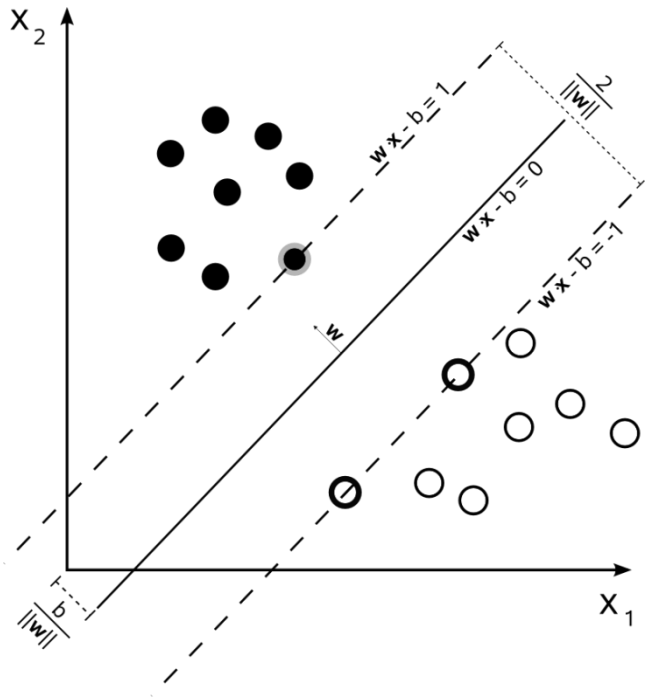

Figure 2.1 Support Vector Machine

The one-class SVM can be considered as a two-class SVM where all the training data are of the first class while the second class is originally only composed of the origin point [11]. The basic idea of the one-class SVM is to map the input data into a high dimensional feature space using a certain kernel function and constructs a decision function to accurately tell the data of one class from the other with the maximum gap.

As SVM is one of the most popular and accurate classifier, there are already a number of implementations, such as looms, Weka [27], TinySVM [28], etc; The one we use in our study is LIBSVM [12], which is developed and maintained by Dr. Chi-Jen Lin's research group in the National Taiwan University. Currently, LIBSVM provides multiple programming languages such as C++, Python, Matlab and Java, which is our choice.

In LIBSVM, There are mainly 3 types of parameters for the SVM classifier: svm type, kernel type and value of C (and Gamma). For the study of rescaling, we use c-svc as the svm type, linear kernel as the kernel type; for the study of ensemble methods, we choose one-class svm as the svm type and rbf kernel as the kernel type. In both study, we use cross validation to decide the value of C (and Gamma), that is, we exhaustively calculate the accuracy for all the possible values of C, and choose the one that produces the highest accuracy [13].

2.2 Bagging

The idea of bagging was first proposed by Leo Breiman in 1994 [14]. It is a machine learning ensemble algorithm that can improve the classification and regression models in terms of stability and classification accuracy. Bagging can also help avoid overfitting by reducing the variance.

The process of bagging is to uniformly sample a subset D_i from the training set D of size n with replacement for m times, build m models with each individual subset of training data, and generate the final prediction of examples using some coordination approach [15].

2.3 Clustering

The goal of clustering is to assign a set of examples into some clusters, so examples in each cluster are more similar to each other than those in other clusters [16].

In our study, we use a bottom-up hierarchical clustering approach. That is, during each iteration, we always merge a pair of clusters whose distance is the shortest among all the candidates. The clustering stops when all the clusters have merged into one [17].

2.4 Rescaling

The theoretical basis of rescaling is that the objective function consists of two terms that compete with each other: (1) the goodness-of-fit (to be maximized), and (2) the number of variables (to be minimized). And the process of rescaling could be represented in the following way:

1. Train a regular linear SVM.

2. Re-scale the input variables by multiplying them by the absolute values of the components of the weight vector w obtained.

3. Iterate the first 2 steps until convergence [18].

The main variation of different implementation of this algorithm lies in step2 that is how to choose the weight vectors from the training model. In our study, we directly make use of the model built by the SVM training process by only allowing the attributes appear in the supporting vectors to be involved in the next iteration; further details of the approach will be discussed in the following chapters.

2.5 N-gram

An n-gram is a contiguous sequence of n items from a given sequence of text or speech. The n-gram model can be used to predict the next item in such a sequence in the form of a (n - 1)-order Markov model, and it is widely used in fields like computational biology, data compression and natural language processing. Two of the advantage of the n-gram model is its simplicity and the ability to scale up [19].

For example, the DNA section: ...AGTCCAGGT... will produce the following sequences: AG, GT, TC, CC, CA, AG, GG, GT when being applied to the 2-gram model; and produce AGT, GTC, TCC, CCA, CAG, AGG, GGT when using 3-gram [20].

2.6 State of Art

Siddiqui et al. used data-mining techniques to detect Trojans [21]. They mined n-grams from the body of Trojans and used these as features. Their dataset they used contains 3000 Trojans and 1722 clean examples. Random Forest and Principal Component Analysis algorithms were used for the purpose of feature selection, and the Random Forest algorithm and SVM for classification. Their method could accurately predict 94% of the new Trojans.

Schultz et al. presented a data-mining framework to detect new executables [22]. They used 4266 programs of which 3265 were malicious and 1001 were clean. They applied three kinds of algorithms: an inductive rule-based learner, a probabilistic predictor, and a multi-classifier. By porting the classification algorithms into a signature-based detection algorithm, 97.76% of the malwares could be detected.

Ye et al. presented an objective-oriented association mining system to detect malware with an accuracy of 92% [23].

Chapter 3
Rescaling

3.1 Basic Information of Data

The original data was provided by the Laboratory for Dependable Distributed Systems at University of Mannheim at the following web site: http://pi1.informatik.uni-mannheim.de/malheur/#appset. In general, there are 24 classes of malwares in total, the name of each class and number of examples could be found in Table 3.1:

Table 3.1 Malware Class Names and Numbers

Malware Class Name	Number of Example
ADULTBROWSER	262
ALLAPLE	300
BANCOS	48
CASINO	140
DORFDO	65
EJIK	168
FLYSTUDIO	33
LDPINCH	43
LOOPER	209
MAGICCASINO	174
PODNUHA	300
POISON	26
PRONDIALER	98
RBOT	101
ROTATOR	300
SALITY	85
SPYGAMES	139
SWIZZOR	78
VAPSUP	45
VIKING_DLL	158
VIKING_DZ	68
VIRUT	202
WOIKOINER	50
ZHELATIN	41

The malware behavior was extracted using CWSandbox. There are 3131 examples in total, and for each example, the research group at University of Mannheim provided 3 formats of

malware behavior description, namely, CWSandbox version and in the MIST encoding version, which is the version we use. A snapshot of the file format is provided in Figure 3.1.

```
1 # process 00000000 0000066a 022c82f4 00000000 thread 0001 #
2 02 01 | 00000000 0000066a 00020000
3 02 02 | 00006b2c 047c8042 000b9000
4 02 02 | 00006b2c 047c8042 00108000
5 02 02 | 00006b2c 047c8042 00091000
6 02 02 | 00006b2c 047c8042 00049000
7 02 02 | 00006b2c 047c8042 000aa000
8 02 02 | 00006b2c 047c8042 00092000
9 02 02 | 00006b2c 047c8042 00011000
10 02 02 | 00006b2c 047c8042 0008b000
11 02 02 | 00006b2c 047c8042 00058000
12 02 02 | 00006b2c 047c8042 0013d000
13 02 02 | 00006b2c 047c8042 0001d000
14 02 02 | 00006b2c 047c8042 0000d000
15 02 02 | 00006b2c 047c8042 00011000
16 02 02 | 00006b2c 047c8042 00091000
17 02 02 | 00006b2c 047c8042 00108000
18 02 02 | 00006b2c 047c8042 00058000
19 02 02 | 00006b2c 047c8042 00028000
20 02 02 | 00006b2c 047c8042 0013d000
21 10 02 | 000040 09f7fa31 00000008 00000000 0000066a
22 10 02 | 000020 09f7fa31 00000008 00000000 0000066a
23 10 02 | 000040 09f7fa47 00000008 00000000 0000066a
24 10 02 | 000020 09f7fa47 00000008 00000000 0000066a
25 10 02 | 000040 07fa3507 00000008 00000000 0000066a
26 10 02 | 000020 07fa3507 00000008 00000000 0000066a
27 10 02 | 000040 0f7fa358 00000008 00000000 0000066a
```

Figure 3.1 Snapshot of the File Format

From the second line until the end of the file, each line is actually a record of a system call made by the malware at run time, the type of the call is encoded into the first two integers in each line, where the first integer is the major operation code while the second is the minor operation code, and according to our experiment, the accuracy obtained by only considering the major code is higher than that when also taking the minor code into consideration. Hence, in the following study, only the major code is used. As there are 20 kinds of operations in total, we could achieve a unique index for each n-gram attribute using the following formula:

$$Index = \sum_{i=1}^{n} X_i * 400^{i-1}$$

Here, X_i is the ith value in a single gram, in other words, the index is calculated as the weighted sum of each individual attribute in the gram. It is not hard to figure out that if a malware has n_1 operations in total, the corresponding n_2-gram training example will have (n_1 - n_2) attributes.

The first experiment we conducted in our study was to find out the relationship between the value of n for n-gram and the accuracy of prediction. To perform this task, we simple use the traditional SVM classifier without any modification to check the different accuracy that could be achieved when using different proportion of training data and value of n.

The process of the experiment could be described in Figure 3.2:

```
for each training percentage in 10%, 30%, 50% do
    for each value of n in [2, 3, 4, 5, 6, 7] do
        double sum = 0;
        for i = 1 : 40 do
            random sample the training data;
            train a model with the training data;
            predict the value of the testing data
            sum += accuracy;
        end
        accuracy = sum / 40;
    end
end
```

Figure 3.2 Process of Deciding Value of n

That is for each possible combination of the parameters, we run the system 40 times and the arithmetic average of the 40 results is used as the final accuracy for each combination.

The result of this process is shown in Table 3.2:

It is easy to come to the conclusion that for any given training percentage, the accuracy decreases monotonically when the value of n increases. As a result of this observation, we stick to the value of 2 in our later settings.

Table 3.2 Result of Different Choices of Training Percentage and Value of n

Training percentage	Value of n	Accuracy
10	2	95.528
10	3	93.802
10	4	92.854
10	5	90.356
10	6	89.562
10	7	88.751
30	2	97.024
30	3	95.652
30	4	94.152
30	5	93.241
30	6	91.478
30	7	89.540
50	2	98.757
50	3	96.823
50	4	95.652
50	5	93.447
50	6	92.548
50	7	90.612

3.2 Multi-grams

Our first attempt to increase the accuracy is to involve multiple choices of n for the n-gram when building the model, with the thought that there is a chance for each gram to make up the deficiency of others [24]. For example, instead of using 2-gram or 3-gram solely, we can train a model with 2-gram and 3-gram simultaneously (noted as 2, 3-gram), take the DNA section example we used in the previous chapter again:

...AGTCCAGGT... will produce the following sequences: AG, GT, TC, CC, CA, AG, GG, GT, AGT, GTC, TCC, CCA, CAG, AGG, GGT when using 2,3 gram, which is the union of sequences produced by 2-gram and 3-gram respectively. However, after we tried to verify the idea with only a few combinations(the result is shown in Table 3.3, the training percentage is 10%), the thought was proved to be wrong, since instead of achieving an accuracy higher than

each individual choice of n, the multi-gram approach cannot work better than the best individual choice, namely, when n = 2.

Table 3-3 Result of Multi-gram Experiment

Combination of Values of n	Accuracy
2, 3	94.074
2, 4	93.323
2, 3, 4	93.252
2, 5	93.137
2, 6	92.772
2, 7	92.034

3.3 Rescaling

3.3.1 Algorithm

As we have mentioned in the introduction to LIBSVM, the model of the SVM generated from the training process is actually stored in a plain text file, a snapshot of which is provided as Figure 3.3; As a result of this, we could easily parse the model generated by the SVM classifier:

```
svm_type c_svc
kernel_type linear
nr_class 2
total_sv 14
rho -0.997508910195926
label 16 7
nr_sv 7 7
SV
3.2144489095463644E-8 275:0.13040442571536293 276:0.03633851284857542 282:0.018838969656054
1.14637128274107 48E-6 275:0.04346814190512098 276:0.0254369589940028 282:0.01883896965605489
4.775524644970103E-8 275:0.19560663857304442 276:0.03088773592128911 282:0.01883896965605489
0.01 306:0.004703027076508521 262:0.0029835505763088944 56:0.004337771719002861 149:0.004424
0.002616977317868601 306:0.004703027076508521 262:0.0029835505763088944 56:0.004337771719002
2.5619938779766174E-8 275:0.08693628381024196 276:0.045423141060719284 129:4.584436031622643
0.007311658575998355 275:0.02173407095256049 306:0.004703027076508521 52:0.00106403571347555
-5.2810703106138 33E-4 275:0.15213849666792342 278:1.4058623304043054E-4 276:0.009084628212144
-4.918537162300482E-4 275:0.10867035476280246 206:0.003338109332676645 338:0.001405636061598
-8.264063411774207E-4 275:0.13040442571536293 276:0.005450776927286314 282:0.018838969656054
-1.9685864747397285E-4 275:0.5216177028614517 276:0.03633851284857542 282:0.0753558786242195
-9.7236403062715 37E-4 275:0.13040442571536293 276:0.005450776927286314 282:0.037677939312109
-0.008722541350920114 275:0.10867035476280246 65:0.00678248545781805 66:0.0761478934134832 2
-0.008191756667333929 275:0.10867035476280246 65:0.00678248545781805 66:0.10153052455131094
```

Figure 3.3 A Snapshot of the SVM Model

The first 7 lines contain the basic information and parameters of the obtained model, such as type, kernel type, number of classes, labels of classes, etc. From the 9th line, every line descripts

a supporting vector used by the model, for example, the 10th line could be parsed as a supporting vector whose weight is approximately $1.14 * 10^{-6}$, and it contains the attribute 275, 276, 282 with the value of 0.043468, 0.025437 and 0.018839 respectively.

For the sake of simplicity, we first focus on the situation where only two classes are involved. However, it is not easy to find such two classes, since even the traditional SVM could make a very accurate prediction on this data set according to the results shown in Table 3.2. To perform this task, instead of simply noting the accuracy of prediction, we also maintain a confusion table to discover the root of misprediction. The confusion table of an execution of the 7-gram execution is shown in Figure 3.4.

Result

Predicted Value

True Value	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1	235	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	258	0	0	0	0	0	0	0	0	0	0	0	0	0	7	0	0	0	0	0	0	0	0
3	0	0	39	0	0	0	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0	0	0
4	0	0	0	93	0	0	0	0	0	0	0	0	0	0	0	32	0	0	0	0	0	0	0	0
5	0	0	0	0	33	0	4	0	0	0	0	0	0	0	0	20	0	0	0	0	0	0	0	2
6	0	0	0	0	0	152	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	11	0	0	0	0	0	0	0	0	16	0	0	0	0	1	0	0	0
8	0	0	0	0	0	0	0	39	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	192	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	131	0	0	0	0	0	23	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	272	0	0	0	0	2	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	8	0	0	0	0	0	0	0	0	18	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	3	0	0	0	0	0	83	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	89	0	1	0	0	0	0	0	1	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	189	82	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	6	0	0	0	0	0	0	0	0	58	0	0	0	0	6	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	0	114	0	0	0	1	6	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	66	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	28	0	0	13	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	144	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	0	0	0	0	51	0	0	0
22	0	0	0	0	0	0	4	4	0	0	0	0	0	0	0	12	3	0	0	0	0	158	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	46	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	35

When using 7 gram and the proportion of training data is 10.0. The corresponding accuracy is 88.75088715400994.

Figure 3.4 Confusion Table

From the above figure, we learned that the classes with label 7 and 16 are the hardest to predict: more than half of the examples of class 7 are predicted as other classes while a lot of examples that do not belong to class 16 are predicted to be of this kind. As a result of this, we use these two classes in our study of rescaling, and we stick to 7-gram since in this case, the result of the traditional training process has the largest potential to be improved.

As mentioned earlier, our main idea of rescaling is to iteratively build a model with the attributes used by the support vectors; by pushing other attributes to 0, the selected attributes are actually rescaled to be larger. The training process will terminate once the accuracy begins to decrease, then we go back to the last iteration, and use this model as the finally trained model.(Figure 3.5):

```
train a model m1 with SVM;
i = 1;
accuracy1 = accuracy gained through m1;
accuracy2 = 0;
while (accuracy2 > accuracy1) do
    Hashtable<Integer, Double> table;
    for each support vector vj ∈ mi with weight wj do
        for each (attribute, value) pk ∈ vj do
            if pk.key ∈ table then
                table.get(pk.key) += wj * pk;
            end
            else
                table.put(pk.key, wj * pk);
            end
            for each training example ej do
                for each (attribute, value) pair pk ∈ ej do
                    if pk.key is ∈ table then
                        pk.value = pk.value * table.get(pk.key);
                    end
                    else
                        delete pk;
                    end
                    train a model mi+1 with the updated training data;
                    accuracy2 = accuracy gained through mi+1;
                    i++;
                end
            end
        end
    end
    accuracy = accuracy1;
end
return accuracy;
```

Figure 3.5 Pseudocode of Rescaling

3.3.2 System Design

(1) Flowchart

After the user input the selected parameters through the user interface (shown in Figure 3.6), the system will begin running. The flowchart of the system, especially with regard to the difference of training process in different methods is shown in Figure 3.7.

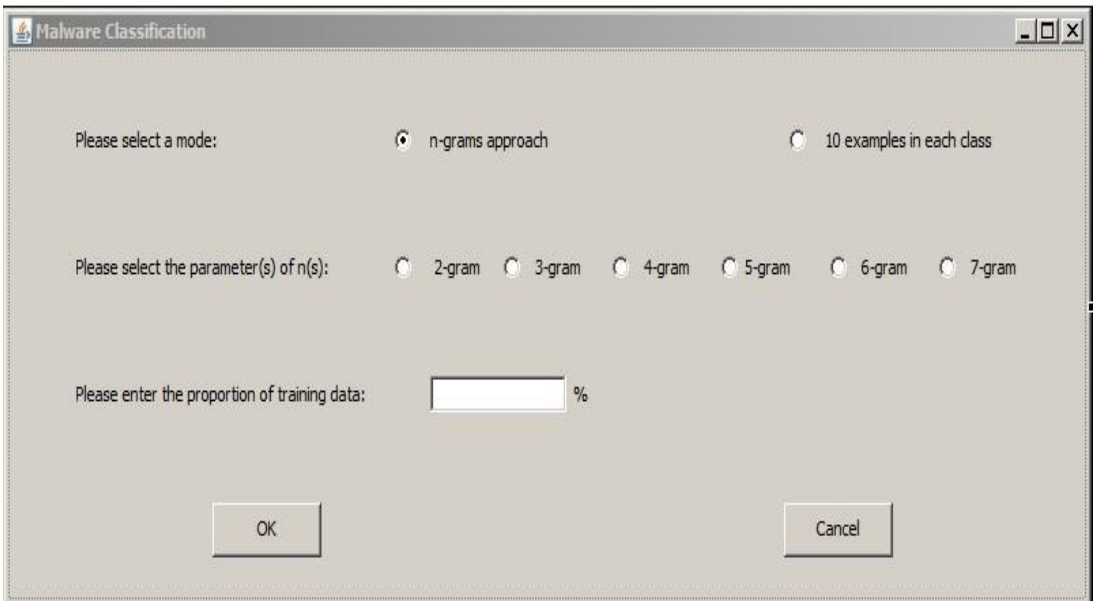

Figure 3.6 User Interface of the Rescaling System

(2) Class diagram

The whole system can be seen as two parts: the SVM classifier contained in the LIBSVM library and the data processing and file manipulator part developed by us. It's nontrivial to mention that although the original LIBSVM package could perform the calculation of classification, we modified it a little to adapt to our needs in the experiment, especially to allow some kinds of return values for the methods.

We mainly designed and implemented 6 classes, there relationships in the form of class diagram are shown in Figure 3.8.

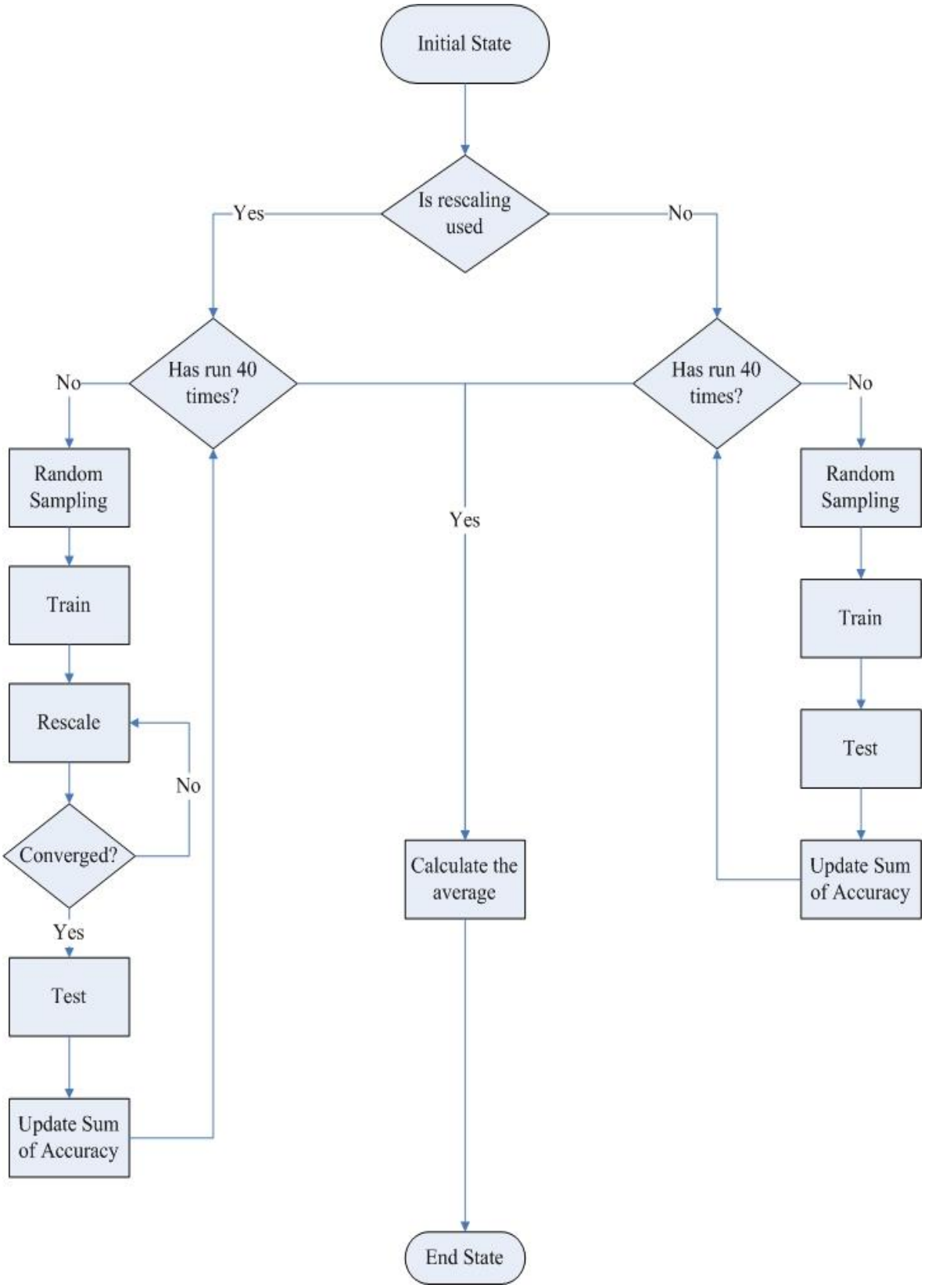

Figure 3.7 Flow Chart of The Rescaling system

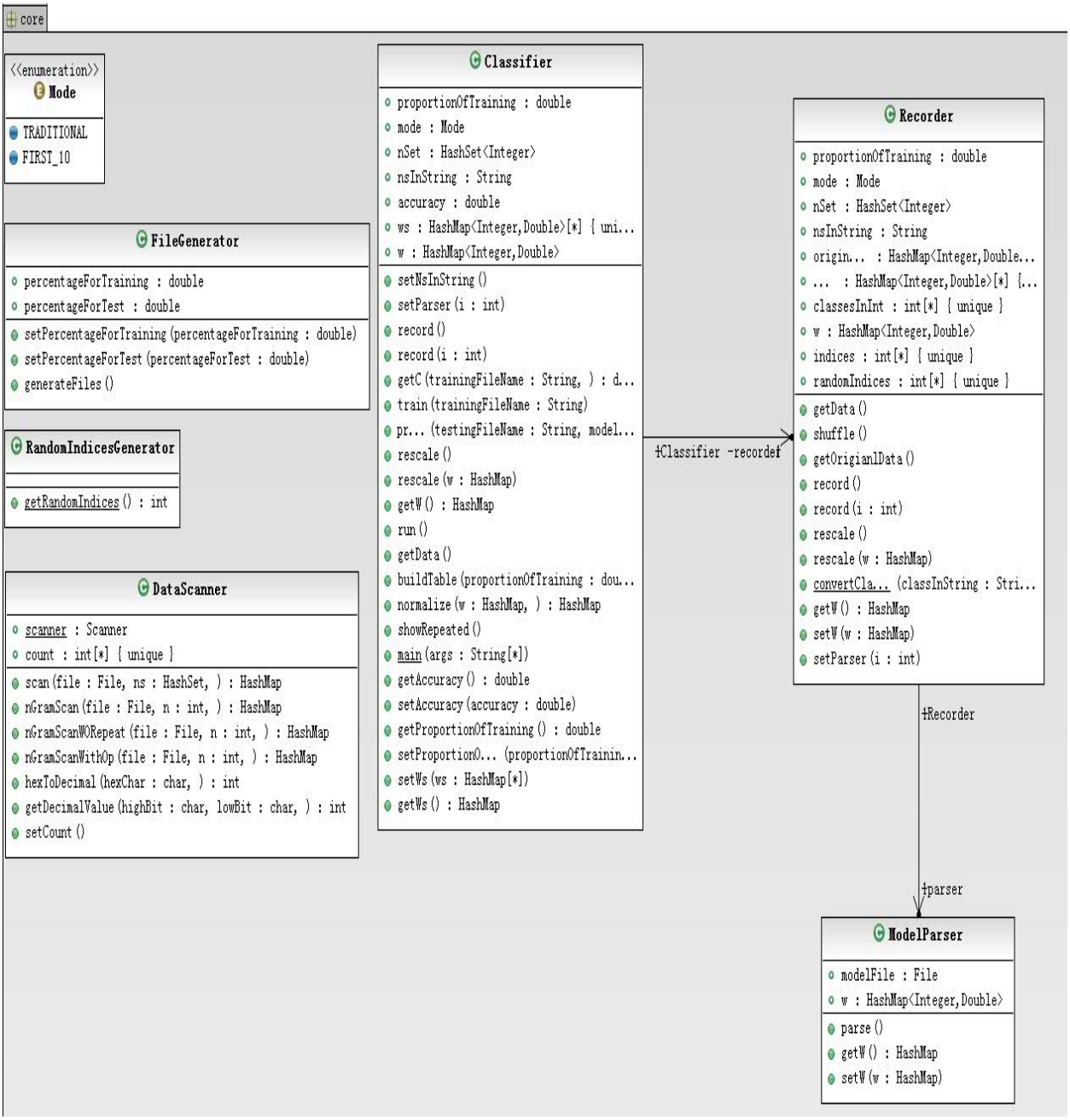

Figure 3.8 A Partial Class Diagram of the Rescaling System

The functions of these classes are:

DataScanner reads the original data file we obtained from the internet using the n-gram rules and translate it into the data file that could be parsed by SVM;

FileGenerator split the data into training data and testing data according to the proportion of traning data;

RandomIndicesGenerator generates the random indices we need to do the random sampling;

Classifier is actually the core of the system, it calls other components of the system to complete the classification job and do the record task;

Recorder calls FileGenerator to update and rewrite the training data during each iteration of the training process.

Model Parser is only used when rescaling is enabled, it parses the text of the SVM model and store the information of the support vectors into a hashtable;

3.3.3 Result and Discussion

We compared the accuracy of the rescaling training method and the traditional training method with training percentage of 10%, 30% and 50%.

For each setting of parameters, the system runs 40 times and the arithmetic average of each running is our final result, which is shown in Table 3-4:

Table 3.4 Comparison between Rescaling and Traditional Training Methods

Training Percentage	Accuracy	
	Traditional	Rescaling
10	68.762	80.265
30	77.053	89.743
50	88.634	96.227

As the two classes we choose in this phase of experiment only have 33 and 84 examples respectively, if we sample the training set by percentage, there is a chance that the number of one class will be too small for training. Instead, we can create a training set of the size 20, with 10 examples from each set. Again, the system runs 40 times and the arithmetic average of each running is our final result, which is shown in Table 3-5:

Table 3.5 Comparison between Traditional and Rescaling Training Method (2)

Method of Training	Accuracy
Traditional	82.423
Rescaling	92.371

From the above experiments, we can learn that:

(1) Rescaling effectively magnifies the useful information contained in the given data and to some extent, could eliminate the noise from the data.

(2) When we do not have enough data for training or there is a large unbalance between the numbers of examples of each class of data, sampling by absolute number works better than sampling by percentage. In our experiment, the second sampling strategy only samples less than 20% of the training data but achieves a higher accuracy than using 30% of the data for training use.

(3) Our experiment shows that rescaling does enhance the effectiveness of training process, however, currently the result is only derived from a certain group of data, and more experiments are needed to further validate the effectiveness of the method.

Chapter 4
Ensemble Methods

We have briefly described the basic concepts of bagging and clustering earlier, here we are providing the detailed implementations of the two methods, experiment settings and the results we got through these work. The dataset we use contains 3683 entries of examples, where 3663 examples are malware (labeled as positive) and 20 are clean (label as negative). As we are focusing on one-class SVM, all the negative examples are preserved as testing data.

4.1 Bagging

Instead of building one model with all the training data, we randomly pick up a fixed percentage of training data and build a model with this subset of training data [14][25]; by repeating this process for certain times, we will achieve a set of models, it is nontrivial to point out that overlapping is allowed between subsets. Later in this chapter, we will discuss how to coordinate these models and predict a testing example. In our study, experiments on bagging was done on the Matlab platform since most of the operations involved in bagging can be transformed to basic mathematical manipulations.

Apart from the training process, we also need to output the training data read by Matlab into plain text file in order to guarantee the accordance of sequence of data: In order to compare the two different ensemble methods, we want to make sure that the result of experiment will not be affected by the variance of the possible training data, that is, in each iteration, the same sample of training data and testing data should be used by bagging and clustering. As we always run the bagging system before the clustering system, this task is performed each time Matlab reads training data from the original data file.

4.2 Clustering

We realize that commercial anti-virus software not only labels software as malicious or normal but also label malwares with proper types that they should belong to. By making use of this observation, we also try to involve the similar idea in our work: we try to group the training examples into clusters, and then we gradually merge smaller clusters into larger ones, build a model if the newly generated cluster is large enough. Finally we will end up with a cluster that contains all the training examples. To be more specific, we maintain a list of clusters. Initially, every training example is a cluster by itself, and in each iteration, we merge the two clusters whose distance is the smallest, delete them from the list and add the new cluster into the list, besides, as long as the size of the new cluster is greater or equal than a preset threshold, we add it to our final list that contains only the clusters we will use to build model with.

The distance between two clusters c_1 and c_2 is defined as:

$$\text{distance}\,(c1, c2) = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} distance(e_i,\ e_j)}{m * n}$$

Here, m is the number of clusters in c_1, n is the number of clusters in c_2,

distance (e_i, e_j) is the distance between two examples e_i and e_j, which can be calculated using the following process (Figure 4.1):

```
Input: ei
Input: ej
int counter = 0;
for each attributes a ∈ ei do
    if a is not ∈ ej then
        counter++;
        for each attributes a ∈ ej do
            if a is not ∈ ej then
                counter++;
            end
        end
    end
end
return counter;
```

Figure 4.1 Calculation of Distance

Based on the above analysis, we can use a greedy approach to merge all the examples into one cluster, that is: in each iteration, we always merge the two clusters whose distance is the shortest into one larger cluster. The idea is shown in Figure 4.2:

```
clusterList = Empty;
finalList = Empty;
for e ∈ training data do
    create cluster c with a single example e;
    add c into clusterList;
end
while clusterList.size > 1 do
    double sDistance = int.max;
    double distance = 0;
    create an empty Cluster cluster1;
    create an empty Cluster cluster2;
    for each cluster c1 ∈ clusterList do
        for each cluster c2 ∈ clusterList do
            for each example e1 ∈ c1 do
                for each examples e2 ∈ c2 do
                    distance += distance(e1, e2);
                end
            end
        end
    end
end
if distance < sDistance then
    sDistance = distance;
    cluster1 = c1;
    cluster2 = c2;
    create an empty cluster cluster3;
    add all the examples in cluster1 into cluster3;
    add all the examples in cluster2 into cluster3;
    add cluster3 into the clusterList;
    delete cluster1 from clusterList;
    delete cluster2 from clusterList;
    if cluster3.size >= threshold
    add cluster3 into finalList;
end
return finaList;
```

Figure 4.2 Pseudocode for Clustering

Unlike bagging, we use Java to implement the system for clustering, since all the data structures involved in the study are provided intrinsically by the Java library.

4.3 Coordination Methods

Both of the techniques will create a bunch of models, each of which will make its own prediction, hence a new problem is aroused—how to make a final decision given these individual predictions. In our work, we have studied the following two coordination approaches:

(1) Voting: The prediction of every single model is equally counted: we maintain a counter whose initial value is 0, and any prediction that labels a testing case as positive increases the counter by 1 while negative predictions decrease it by 1; after calculating all the predictions, if the counter is positive, then the testing case is positive, and vice versa. The corresponding flowchart is provided in Figure 4.3.

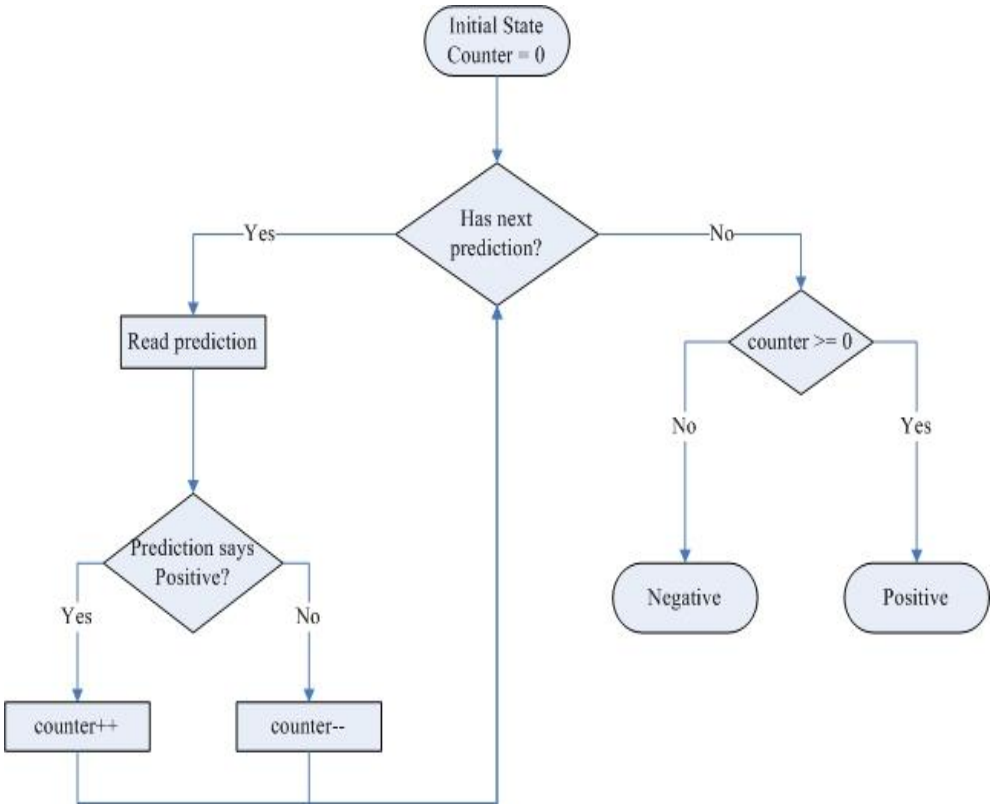

Figure 4.3 Mechanism of Voting

(2) One positive to be positive: as the name suggests, as long as any one of the built models predicts an entry of testing data as positive then we label it as positive. It's not hard to

learn that this approach will effective lower down the chance of false-negative error but could result in more false-positive error. Again, a flowchart (Figure 4.4) is plotted to help understanding the idea.

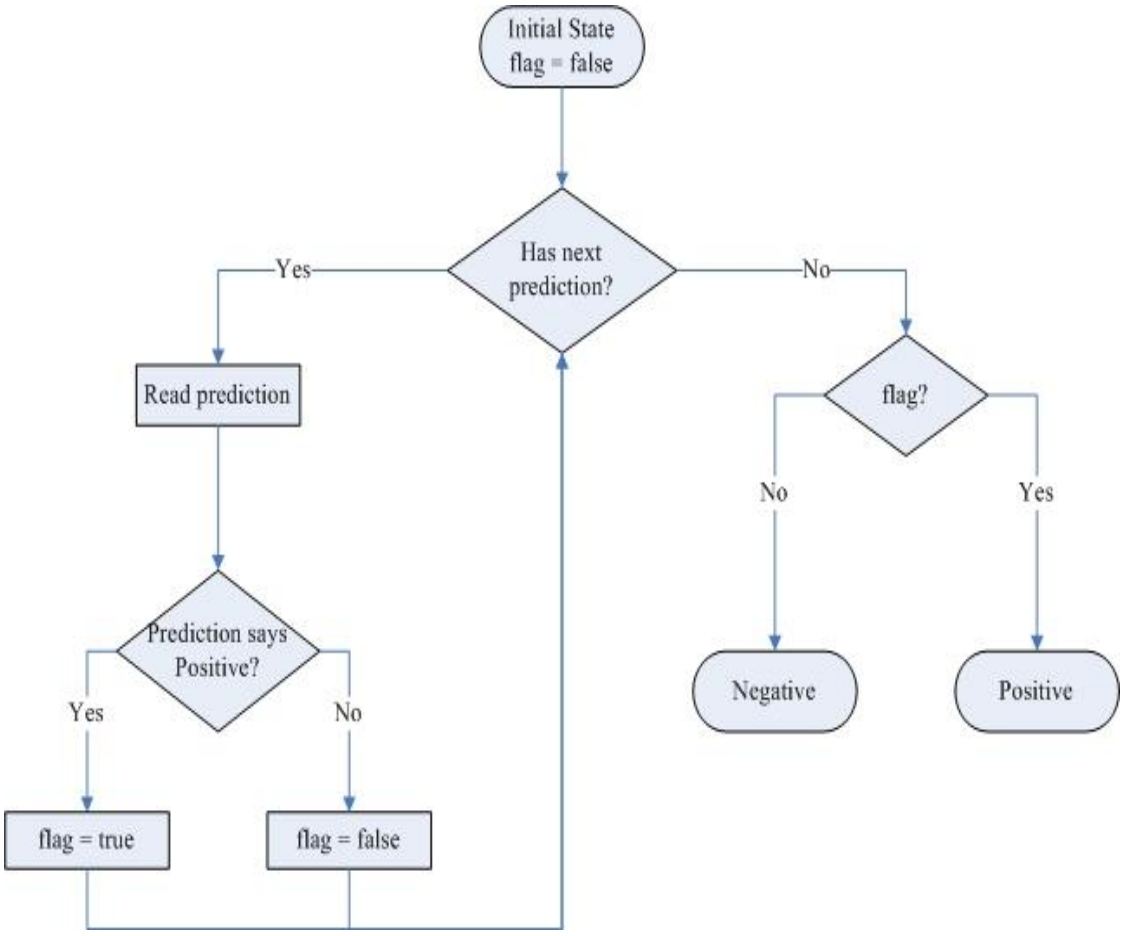

Figure 4.4 Mechanism of the One Positive to Be Positive Approach

By combining the two techniques and labeling strategies, there are four different approaches to the problem, and in our work, together with the traditional learning process, we have compared all these five different approaches:

(1) Build a model directly according to the training data.

(2) Training with bagging and labeling using voting.

(3) Training with bagging while one positive to be positive.

(4) Training with clustering and labeling using voting.

(5) Training with clustering while one positive to be positive.

4.4 Parameters Settings

In our experiment, apart from the parameters for LIBSVM we discussed earlier, we have four parameters for the training process:

(1) Number of bags: this parameter indicates how many bags are created during the training process; possible values of this parameter are 20, 40, 60, 80, and 100.

(2) Number of training examples: the number of examples to be used in the training process. Possible values are inclusively between 50 and 300, with an interval of 50. Given the fact that there are 3383 examples in total, the percentage of training is strictly less than 10%, which is able to simulate the condition that there may not be enough training samples.

(3) Percentage of bagging: what proportion of training samples are used in each bag. Values for this parameter are 70%, 80% and 90%.

(4) Threshold for clustering: this value indicates the minimum size of a cluster that could be added into the final list of cluster that will be used to build a model.

The process of our experiment can be describes as:

(1) Transform the information of malwares and normal softwares that are initially binary into .mat files that could be processed by Matlab.

(2) Perform the (1) ~ (3) approaches described above using the Matlab implementation of SVM with all possible combinations of the 3 parameters we mentioned previously. In order to guarantee the consistency of training data, we also output the sequence of training data and testing data of each execution for the next step. The parameters for libsvm are: one-class for the type and radial basis function (rbf) for the kernel type.

(3) Perform the (4) ~ (5) approaches using the Java implementation of SVM with 50 and 100 training examples, thresholds of 20%, 40%, 60% and 80% of the total number of training examples respectively, the input is given by the sequence generated in the last step.

The flowchart (Figure 4.5) for the above description is provided below:

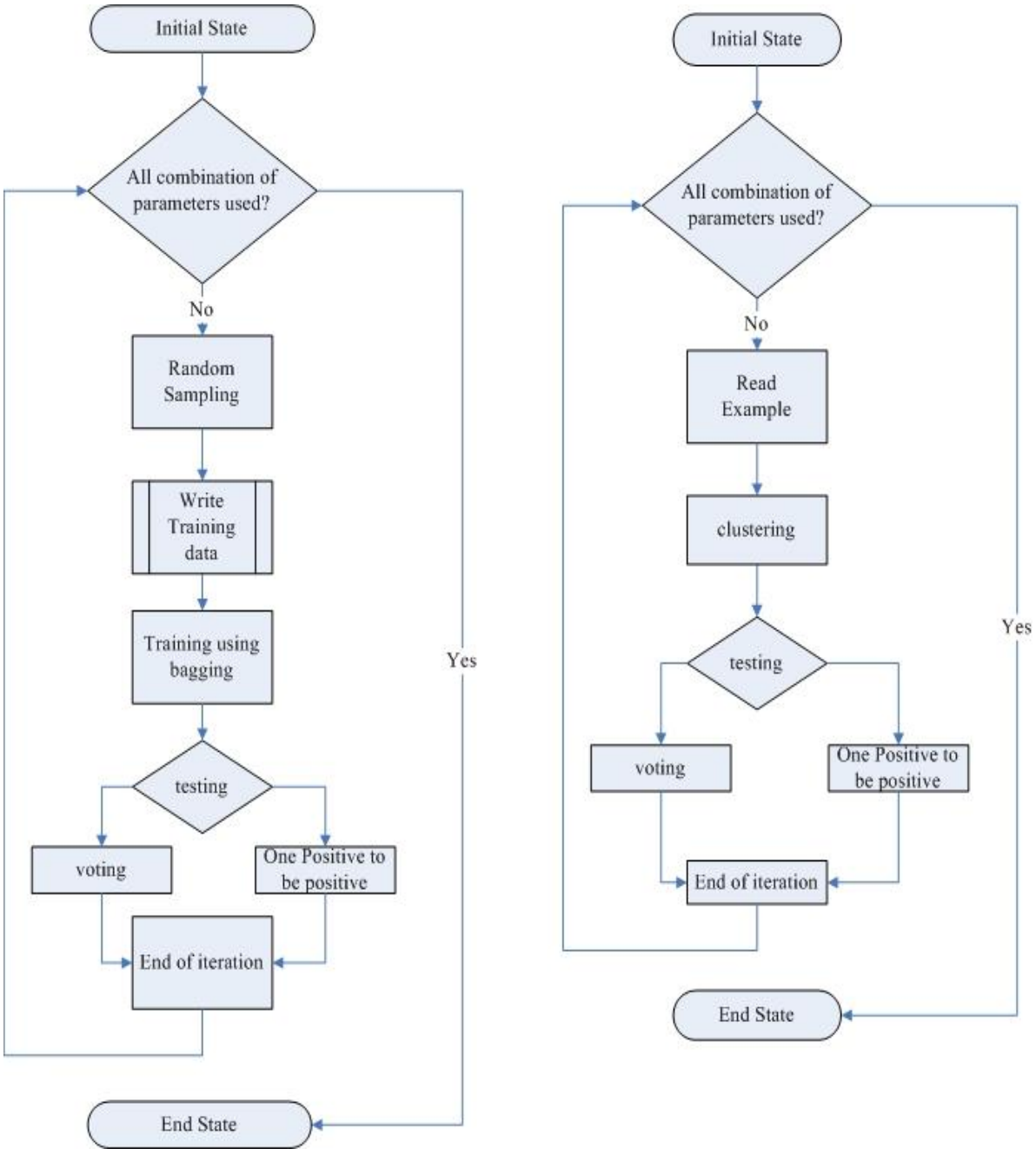

Figure 4.5: Flowchart of study of Bagging (left) and Clustering (right)

4.5 System Design

4.5.1 Class Design

Here, we only provide the design sketches of the clustering system since the bagging part is conducted on Matlab where only scripts are used. Apart from the classes built in the libsvm library, we have the following 5 classes in the class diagram (Figure 4.6).

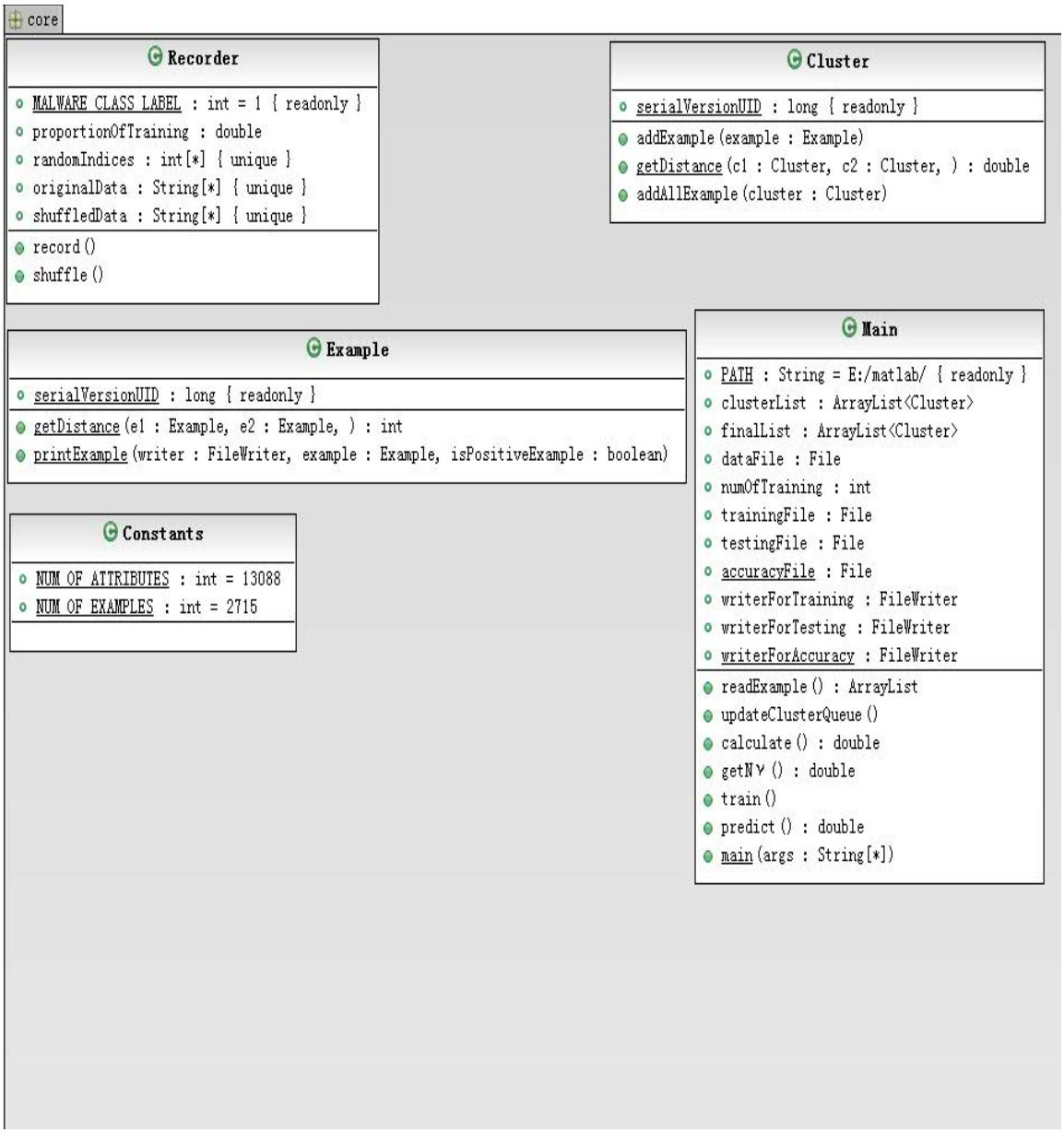

Figure 4.6 Class Diagram of the Clustering System

The functions of the 5 classes are:

The Recorder class reads the text file that contains the training data written by Matlab and transform the file into the format that could be read by libsvm.

(1) The Example class simulates an example of data, it implements the method to calculate the distance between two examples.

(2) The Cluster class simulates a cluster of example, it implements all the behavior of the cluster and clustering process.

(3) The Main class is the core of the system, it calls the above modules together with the libsvm library to conduct the experiment; it also write the result of each iteration of execution into the file for future use.

(4) The Constants class is a utility that records the unmodified constants of the system; namely, the number of examples and the number of attributes.

4.5.2 I/O Cost

As a well-known principle in computer science, I/O operation is often expensive. Hence we should always try to minimize it. In our study, we cannot decrease the necessary I/O manipulation brought by reading and writing data, but we can try to minimize the cost involved in the communication between the two phases of the experiment. With this goal in mind, together with the fact that the original training data is actually a sparse matrix, we came to the method that only the non-zero attributes of examples should be written into the plain text by Matlab as the training data for clustering. The above process is shown in Figure 4.7:

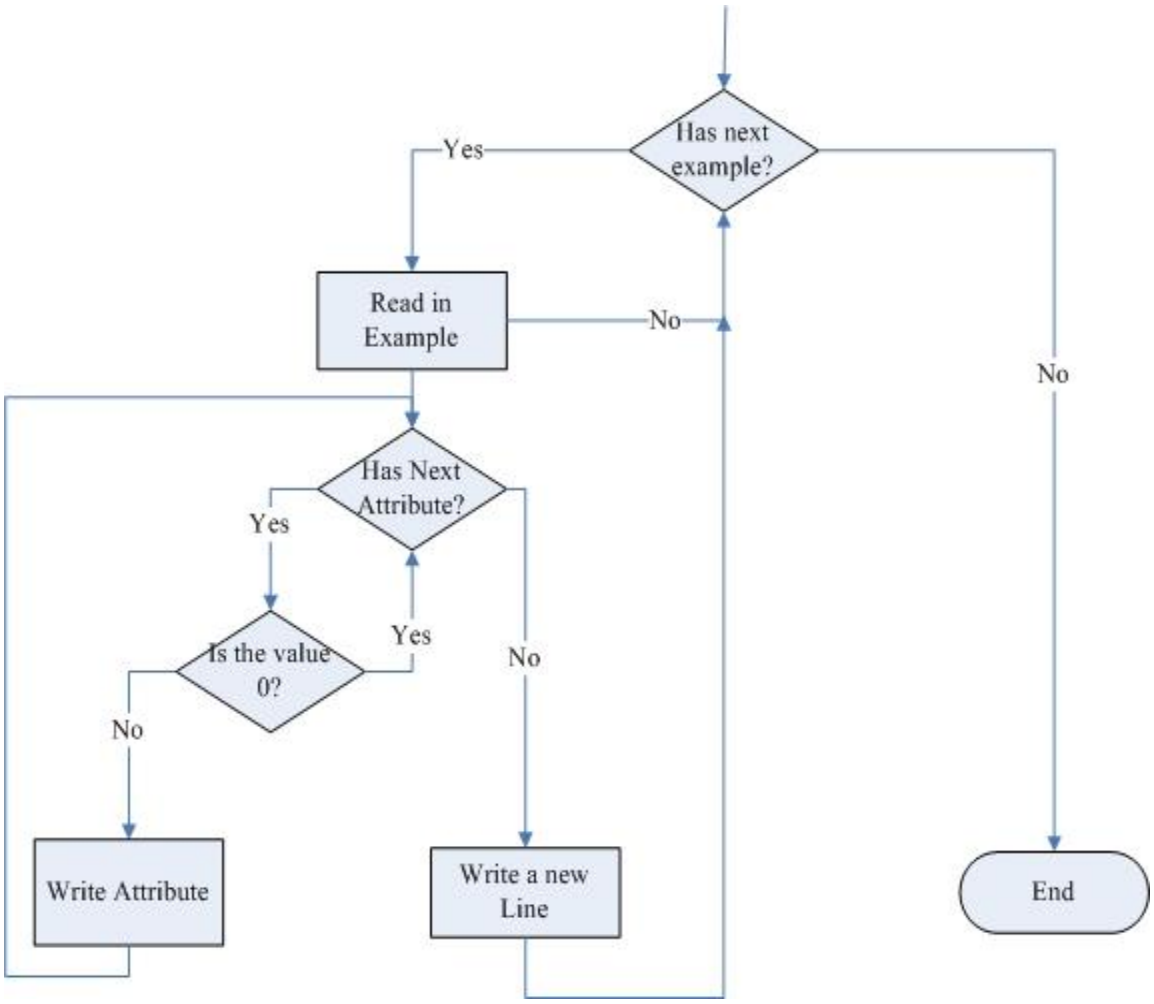

Figure 4.7 Process of writing training data

4.6 Result and Discussion

The result of bagging and clustering is shown in Table 4.1 and Table 4.2 respectively:

As mentioned earlier, the bagging part was conducted first on the matlab platform with the help of the libsvm package, then output the training data and testing data sequentially into plain text files; after that, the clustering part was made on the Java platform by parsing the text files and calling the svm algorithm to perform the calculation. The result of both parts are in the form of accuracy of prediction.

To guarantee consistency, we maintain the same parameter settings for both experiments, namely, we use the same number of training examples, percentage of training and coordination methods in both situation. Consequently, we could derive a very direct understating of their performances in our context.

Table 4.1 Result of bagging and Comparison with the Traditional Approach

# of bag	# of train	% of bagging	Traditional		bagging			
			False Negative	False Positive	One positive to be positive		Voting	
					False Negative	False Positive	False Negative	False Positive
20	50	70	0.170426	0.050000	0.074869	0.050000	0.408898	0.050000
20	50	90	0.173595	0.050000	0.079255	0.050000	0.402837	0.050000
20	100	70	0.130817	0.097500	0.078136	0.050000	0.191440	0.050000
20	100	90	0.120516	0.050000	0.065114	0.050000	0.135069	0.050000
20	200	70	0.058345	0.097500	0.022192	0.145000	0.092882	0.050000
20	200	90	0.071686	0.142500	0.021152	0.147500	0.072567	0.050000
20	300	70	0.036679	0.097500	0.016176	0.477500	0.048231	0.050000
20	300	90	0.020726	0.060000	0.011418	0.152500	0.032233	0.060000
40	50	70	0.212109	0.050000	0.071810	0.050000	0.466468	0.050000
40	50	90	0.198671	0.050000	0.066634	0.050000	0.397191	0.050000
40	100	70	0.118861	0.050000	0.054042	0.100000	0.164875	0.050000
40	100	90	0.116756	0.050000	0.054701	0.097500	0.135897	0.050000
40	200	70	0.030884	0.145000	0.012388	0.387500	0.084204	0.050000
40	200	90	0.072322	0.097500	0.018409	0.145000	0.067153	0.050000
40	300	70	0.021618	0.050000	0.007196	0.657500	0.030672	0.050000
40	300	90	0.023476	0.145000	0.012266	0.145000	0.036411	0.107500
60	50	70	0.208276	0.047500	0.057265	0.050000	0.417465	0.050000
60	50	90	0.224177	0.050000	0.065513	0.050000	0.405563	0.050000
60	100	70	0.142057	0.050000	0.060904	0.050000	0.197193	0.050000
60	100	90	0.123940	0.050000	0.050954	0.050000	0.134143	0.050000
60	200	70	0.042997	0.192500	0.020214	0.287500	0.090990	0.097500
60	200	90	0.091366	0.050000	0.025744	0.050000	0.080075	0.050000
60	300	70	0.024978	0.052500	0.007434	0.810000	0.039102	0.050000
60	300	90	0.018867	0.145000	0.009709	0.257500	0.028115	0.112500
80	50	70	0.204802	0.050000	0.069582	0.050000	0.517963	0.045000
80	50	90	0.240368	0.050000	0.062068	0.050000	0.445931	0.050000
80	100	70	0.133764	0.050000	0.054252	0.050000	0.180382	0.050000
80	100	90	0.163458	0.050000	0.055571	0.097500	0.154070	0.050000
80	200	70	0.040658	0.050000	0.022004	0.135000	0.103018	0.050000
80	200	90	0.086457	0.192500	0.013067	0.240000	0.069275	0.050000
80	300	70	0.020146	0.092500	0.006587	0.745000	0.046209	0.060000
80	300	90	0.020056	0.145000	0.008043	0.145000	0.032337	0.050000

Table 4.2 Result of Clustering

Threshold	number of training	clustering			
		One positive to be positive		Voting	
		False Negative	False Positive	False Negative	False Positive
20	50	0.103312	0.050000	0.565217	0.050000
20	100	0.056879	0.050000	0.495378	0.050000
20	200	0.039573	0.050000	0.456505	0.050000
20	300	0.035127	0.050000	0.425244	0.050000
40	50	0.100371	0.050000	0.466533	0.050000
40	100	0.056988	0.050000	0.391410	0.050000
40	200	0.039946	0.050000	0.347988	0.050000
40	300	0.035084	0.050000	0.322273	0.050000
60	50	0.104034	0.050000	0.398762	0.050000
60	100	0.059326	0.050000	0.329310	0.050000
60	200	0.039269	0.050000	0.288754	0.050000
60	300	0.034733	0.050000	0.263793	0.050000
80	50	0.102442	0.050000	0.208747	0.050000
80	100	0.057985	0.050000	0.152316	0.050000
80	200	0.039511	0.050000	0.135846	0.050000
80	300	0.034633	0.050000	0.117286	0.050000

From the above result, we can conclude that:

(1) As is in accordance with a general principle in Machine learning, more training example will lead to higher accuracy.

(2) Although occasionally suffer from a higher chance of false-positive error, one positive to be positive approach performs better than voting in general.

(3) Higher threshold for clustering will result in higher accuracy for voting.

(4) In general, bagging performs better than clustering in dealing with false-negative error while clustering wins when it comes to false-positive error.

(5) As we only have 20 negative examples (clean software), but have over 3000 positive examples (malware), if we assign the same weight for false-negative and false-positive

errors, we can come to the conclusion that bagging with one positive to be positive coordination method produces the best accuracy among all the 5 approaches.

(6) We need to verify our idea with more negative examples in our future work when they are available.

Chapter 5
Conclusion and Future Work

In general, we have studied the effect of two approaches of learning on the topic of malware classification: rescaling and ensemble methods with one-class SVM:

Rescaling is an iterative process the idea of which is to selectively magnify some of the attributes of the training data and squeezing others to discover the useful information contained in the training data set and filter the noise-like interruption. In our implementation, we select features via parsing the SVM model and we showed that rescaling can effectively improve the accuracy of prediction model.

Both bagging and clustering can reduce the error rate of false-negative error without or with trivial rise of chances of false-positive error in the context of one-class SVM if appropriate coordination method is applied with them, especially when the amount of training data is very limited.

Future work can include: verify our ideas with more sets of data; perform experiments with more ensemble methods; integrate all the experiment environments into one platform.

Bibliography

[1] http://en.wikipedia.org/wiki/Malware

[2] Eric MJolsness, David H.Sharp, Bradley K. Alpert. *Scaling, Machine Learning, and Genetic Neural Nets*. Advances in Applied Mathematics 10, 137-163 (1989)

[3] http://pi1.informatik.uni-mannheim.de/malheur/#appset

[4] David Opitz, Richard Maclin. *Popular Ensemble Methods: An Empirical Study*. Journal of Artificial Intelligence Research 11 (1999) 169-198

[5] Yunqiang Chen, Xiang Zhou, Thomas S. Huang. *One-class SVM for Learning in Image Retrieval*. IEEE Int'l Conf. on Image Processing 2001

[6] Eric Bauer, Ron Kohavi. *An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants*. Machine Learning, vv, 1–38 (1998)

[7] http://robotics.stanford.edu/~ronnyk/glossary.html

[8] Christopher J.C. Burges. *A Tutorial on Support Vector Machines for Pattern Recognition.* Kluwer Academic Publishers, Boston.

[9] Simon Tong, Daphne Koller. *Support Vector Machine Active Learning with Applications to Text Classification.* Journal of Machine Learning Research (2001) 45-66

[10] http://www.cs.cornell.edu/courses/cs578/2006fa/slides_sigir03_tutorial-modified.v3.pdf

[11] Hwanjo Yu. *SVMC: Single-Class Classification with Support Vector Machines*. International Joint Conferences on Artificial Intelligence 2003

[12] C.-C. Chang and C.-J. Lin. *LIBSVM : a library for support vector machines*. ACM Transactions on Intelligent Systems and Technology, 2:27:1--27:27, 2011.

[13] R.-E. Fan, P.-H. Chen, and C.-J. Lin. *Working set selection using the second order information for training SVM.* Journal of Machine Learning Research 6, 1889-1918, 2005.

[14] Breiman, Leo (1996). *Bagging predictors.* Machine Learning 24 (2): 123–140.

[15] LIU, Y., and X. YAO, 1999. *Ensemble Learning via Negative Correlation*. Neural Networks, 12(10), 1399–1404.

[16] R. Sibson (1973). *SLINK: an optimally efficient algorithm for the single-link cluster method.* The Computer Journal (British Computer Society) 16 (1): 30–34.

[17] D. Defays (1977). *An efficient algorithm for a complete link method*. The Computer Journal (British Computer Society) 20 (4): 364–366.

[18] Isabelle Guyon, Andre Elisseeff. *An Introduction to Variable and Feature Selection.* Journal of Machine Learning Research 3 (2003) 1157-1182.

[19] Peter Náther. *N-gram based Text Categorization*. Diploma thesis. Faculty of Mathematics, Physics and Informatics, Institue of Informatics, Comenius University

[20] Z. Volkovicha , V. Kirzhnerb, A. Bolshoyb et al. *The Method of N-grams in Large-scale Clustering of DNA Texts*. Pattern Recognition Society. 2005

[21] M. Siddiqui, M. C. Wang, and J. Lee, *Detecting trojans using data mining techniques.* in IMTIC, ser. Communications in Computer and Information Science, D. M. A. Hussain, A. Q. K. Rajput, B. S. Chowdhry, and Q. Gee, Eds., vol. 20. Springer, 2008, pp. 400–411.

[22] M. G. Schultz, E. Eskin, E. Zadok, and S. J. Stolfo. *Data mining Methods for Detection of New Malicious Executables*. IEEE Symposium on Security and Privacy. Washington, DC, USA: IEEE Computer Society, 2001.

[23] Y. Ye, D. Wang, T. Li, and Ye. *Imds: Intelligent malware detection system*. Proccedings of ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD 2007), 2007.

[24] Dou Shen, Jian-Tao Sun, Qiang Yang, and Zheng Chen. *Text Classification Improved Through Multigram Models.* CIKM '06: Proceedings of the 15th ACM international conference on Information and knowledge management.

[25] Aslam Javed A, Popa Raluca A. and Rivest Ronald L. (2007). *On Estimating the Size and Confidence of a Statistical Audit*. Proceedings of the Electronic Voting Technology Workshop (EVT '07), Boston, MA, August 6, 2007.

[26] Jason Weston. *Leave-One-Out Support Vector Machines*. International Joint Conferences on Artificial Intelligence.Vol.2.1999.

[27] Mark Hall, Eibe Frank, Geoffrey Holmes, et. al. *The WEKA Data Mining Software.* SIGKDD Explorations. Volume 11, Issue1.

[28] http://chasen.org/~taku/software/TinySVM/

Vita

Xing An was born and raised up in Wuhan, China. He received his bachelor's and master's degree in engineering from Wuhan University in 2008. Thereafter, he came to LSU and began to work under the supervision of Dr. Jian Zhang with research focus on machine learning. He will receive his master's degree in May 2013 and plans to continue his study towards his doctorate upon graduation.