Feature Reduction to Speed Up Malware Classification

Veelasha Moonsamy, Ronghua Tian, and Lynn Batten

Deakin University, School of Information Technology,
Melbourne, Australia
{v.moonsamy,rtia,lmbatten}@deakin.edu.au

Abstract. In statistical classification work, one method of speeding up the process is to use only a small percentage of the total parameter set available. In this paper, we apply this technique both to the classification of malware and the identification of malware from a set combined with cleanware. In order to demonstrate the usefulness of our method, we use the same sets of malware and cleanware as in an earlier paper. Using the statistical technique Information Gain (IG), we reduce the set of features used in the experiment from 7,605 to just over 1,000. The best accuracy obtained in the former paper using 7,605 features is 97.3% for malware versus cleanware detection and 97.4% for malware family classification; on the reduced feature set, we obtain a (best) accuracy of 94.6% on the malware versus cleanware test and 94.5% on the malware classification test. An interesting feature of the new tests presented here is the reduction in false negative rates by a factor of about 1/3 when compared with the results of the earlier paper. In addition, the speed with which our tests run is reduced by a factor of approximately 3/5 from the times posted for the original paper. The small loss in accuracy and improved false negative rate along with significant improvement in speed indicate that feature reduction should be further pursued as a tool to prevent algorithms from becoming intractable due to too much data.

Keywords: dynamic analysis, feature reduction, malware classification.

1 Introduction

Malicious software classification supports the products of anti-virus vendors and so is important to the computer industry. In the last several decades, many papers have appeared demonstrating varying approaches to such classification. The papers [12, 17, 5] and the references therein provide the reader with a sample of such work. In all cases, statistical analysis methods are applied and the accuracy of the resulting classification is measured.

In choosing features to use as input to the analysis, the usual approach is to use as many as possible. However, some schools of thought argue that, in general, only a small percentage of available features are needed to provide good classification [1, 6]. The authors of [6] argue as follows: 'In a great variety of fields … the input data are represented by a very large number of features, but only few of them are relevant for predicting the label. In addition, many algorithms become computationally intractable

P. Laud (Ed.): NordSec 2011, LNCS 7161, pp. 176–188, 2012.

when the dimension is high. On the other hand, once a good small set of features has been chosen, even the most basic classifiers … can achieve desirable performance. Therefore, feature selection, i.e. the task of choosing a small subset of features which is sufficient to predict the target labels well, is critical to minimize the classification error. At the same time, feature selection also reduces training and inference time and leads to better data visualization, reduction of measurement and storage requirements.'

The aim of our work is to show empirically that a significant reduction in the number of features is possible while at the same time maintaining a good level of accuracy. We use as a benchmark, the work in [12] which classified a set of 1368 executable samples with an accuracy of 97.4% and correctly distinguished clean from malicious files with an accuracy of 97.3%.

Based on the same set of executables, as a means of significantly reducing the speed at which the test executes, in this paper, we identify a significantly smaller feature set which produces close to the same levels of accuracy as those in [12]. In order to choose an appropriate set likely to retain accuracy, we use Information Gain [16] which is a statistical method discriminating between important and less important features across homogeneous data sets. We are able to reduce running time of the tests by a factor of approximately 3/5.

In Section 2, we review the literature in this area. In Section 3, we describe our experimental set-up. In Section 4, we describe the classification model and in Section 5, we analyse the results. In the final section, we draw our conclusions.

2 Literature Review

Although a new malware variant is disguised, its underlying structure still remains the same and retains the target of propagating the infection. Therefore, the features that appear frequently within a known malware can be used to identify and distinguish between cleanware and malware. Below, we present a few examples of the recent work conducted in the area of feature selection and reduction for malware.

Komashinskiy and Kotenko [4] test their feature selection and reduction technique on 5854 malicious files and 1656 cleanware files. They collect 65,536 features through static analysis and apply Information Gain (IG) to differentiate between most and least important features. The authors then form five different size feature sets which contain 50, 100,150, 200, 250 features respectively and use them to train the classifiers in WEKA [15]. The set of 250 features gives the best classification accuracy of 98%. Using a similar feature reduction method, Wang et al. [13] used IG and gain ratio to decrease the number of features from 1656 to 645, a 60% reduction. The authors used a dataset comprising of 1908 clean files and 7863 malicious files, and obtained an average classification accuracy of 95%.

Mehdi et al. [9] develop a framework that can analyze and detect in-execution malware. They experiment on a set of 100 Linux malware files, collected from a publicly available online database, and 180 cleanware files. Their methodology uses dynamic analysis together with the n-grams method. The authors claim that they present a novel

idea by introducing a component known as a 'Goodness Evaluator' into their framework to reduce the feature set. After classification, the authors conclude that the dataset from 6-grams outperforms that from 4-grams. They also test their classification method against four other algorithms, and obtain a best accuracy of 77%.

The authors of [3, 7, 8, 14] also apply various feature reduction approaches in their respective work to contribute towards malware detection and classification.

3 Experimental Setup

3.1 Data Preparation

In order to benchmark our work against that of [12], we perform the same experiments on the same dataset with the exception that we apply Information Gain to reduce the feature set. We reduce the feature sets separately for each of the tests, malware versus cleanware and malware family classification. IG analyzes the API calls and their parameters separately and examines their effect on the classification results.

For the purpose of their work, Tian et al. [12] executed the data set, comprising 1368 malware files and 456 clean files, for 30 seconds and obtained 1824 log files. To ensure that we were not missing any important features, using the same data set, we let each executable run for 60 seconds and obtained 12 additional log files. Upon investigating the contents of the additional files, we found no relevant additional information. Hence, it can be deduced that a further 30 seconds does not impact on the experiments in any way and so we used the same 1824 log files as in [12] and stored these in our database.

3.2 Data Pre-processing

We first introduce some terminology that we will refer to as we progress through the explanation of our experiments.

An '***APIString***' is made up of an API call together with its corresponding parameters. Below are two examples of APIStrings from our database.

- *RegSetValueExW, 0x18c, IntranetName, 0100000*
- *CreateFileW, \\.\MountPointManager*

The components *RegSetValueExW* and *CreateFileW* are API calls while the remaining segments are parameters.

We use the term '***String***' to refer either to an API call or to a parameter. Thus, using the above excerpt of the log file presented in bullet points, we consider the following items: *'RegSetValueExW', '0x18c', 'IntranetName', '0100000', 'CreateFileW' and '\\.\MountPointManager'* to be Strings.

To have a consistent comparison, we conduct the same two experiments as in [12], malware versus cleanware detection and malware versus malware family classification.

3.3 Feature Extraction and Selection

We make use of a database management system program because of its usability and integrated management interface. We store the features to be used in the test as shown in Table 1 along with a description of the purpose of each field.

Table 1. Strings Database Scheme

Column Name	Description
FileID	Unique file identification number which is generated during execution.
Family	File's family name.
String	Stores the content of String extracted from the file.
Type	Identifies whether the String is an API call or parameter
FreqofStringInFile	Total number of times a String appears within a file.

The number of times a String occurs in the feature set is indicative of its importance in the classification process because the more often it appears in a given malware or cleanware family, the higher the probability that it differentiates itself from the features belonging to other families.

Thus, our next step is to derive a feature selection method and apply it to the String data to select the most significant features. There are two standard approaches to this task, one known as the filter method and the other as the wrapper method ([2], p. 141). The filter method is more appropriate to our data set because it uses an evaluation function which relies on the distance metric. In other words, the most desirable attributes have a greater difference among the entire feature set. While there are several sophisticated feature selection methods available, such as Information Gain, Chi-Square, Fisher's Score, n-grams [8], we choose IG as it has a high adoption rate and is well understood. Some preliminary tests using three feature selection algorithms indicated that IG selected a more comprehensive set of highly weighted features. Moreover, the IG selection method is available to us in the WEKA set of algorithms from which we draw the classifiers for later use in the test.

IG was chosen as it is regarded as a powerful technique for discriminating between important and less important features across homogeneous data such as ours.

Information Gain evaluates the importance of an attribute by measuring the IG value with respect to the class. An example of calculating IG values is provided by [1]: "For a given attribute X and a class attribute Y ∈ {Cleanware, Malware}, the uncertainty is given by their respective entropies H(X) and H(Y)". Then the IG of X with respect to Y is given by IG(Y;X):

$$IG(Y; X) = H(Y) - H(Y|X).$$

We begin by selecting the distinct Strings in each family, as shown in Table 2 and use the numbers in the third column of the table to apply feature selection for two tests.

Table 2. Number of Distinct Strings per Family

Type	Family	Number of Files per Family	Number of distinct Strings
Malware	Agobot	340	1280
	Alureon	58	1375
	Bambo	70	884
	Beovens	144	1022
	Boxed	366	1268
	Clagger	47	1095
	Emerleox	78	1051
	Looked	67	774
	Robknot	119	1145
	Robzips	82	905
	SUB TOTAL	*1371*	
Cleanware	Clean	465	998
	TOTAL	1,836	11797 Strings (with repeats)

3.3.1 Malware Versus Cleanware (M/C)

In this test, we ignore the malware families and treat the set of malware files as a single set. It can be noted from Table 2 that the numbers of malware and cleanware executable files are not proportionate. To deal with this imbalance, we follow the recommendations of [10] and split the total malware executable files into groups of same size as that of the cleanware.

After merging all the malware executables, we obtain a total of 1371 files. We then proceed to build groups of 465 malware files, which is the number of files present in the cleanware group. After generating the first 2 groups, $mGroup_1$ and $mGroup_2$, we are left with 441 malware executable files. To build the third group, $mGroup_3$, we then randomly select an additional 24 (465-441) files from the first two groups, as shown in Figure 1.

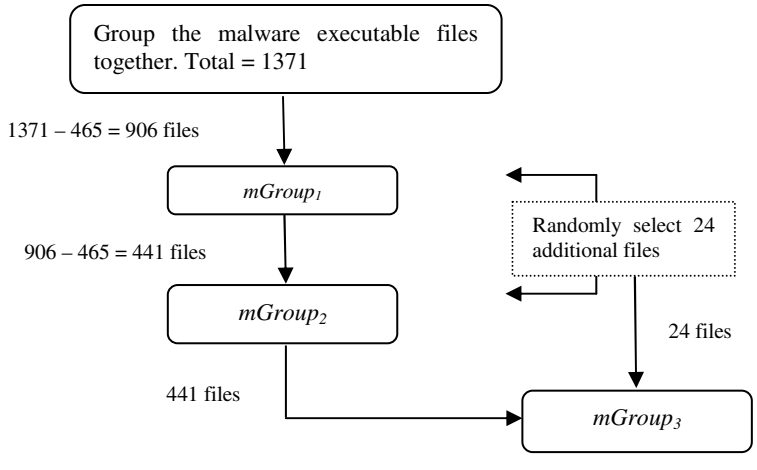

Fig. 1. Generate Malware Groups

We then take each $mGroup_i$ and select the distinct Strings in that particular group. Next, we use the list of distinct Strings to generate a vector for each file within the current $mGroup_i$, where an element within a vector represents the presence or absence of a particular String from the distinct String list. Below is an example on how to generate the vectors.

Suppose distinct String list includes 10 Strings, as shown below:

distinctStringList =
{GetProcAddress,
0x77e60000,
LoadLibraryA,
ExitProcess,
LoadLibraryExW,
ADVAPI32.dll,
0x77dd0000,
RegCloseKey,
WININET.dll,
InitializeCriticalSectionAndSpinCount}

Assuming that file ***F*** is made up of 3 Strings (shown as italicized) from the above list, therefore:

File *F* =
{LoadLibraryA,
ExitProcess,
LoadLibraryExW}

Since the feature vector for ***F*** is made up of the frequency of each String within the file, then the vector for file ***F*** might look as follows:

Vector_F = (0,0,6,3,16,0,0,0,0,0),
where 0 represents the absence of a String and any other number denotes the frequency of that particular String in the file.

We repeat this process for *mGroup1*, *mGroup2* and *mGroup3* to generate the ARFF files and input them into WEKA for feature selection by IG.

3.3.2 Family by Family (F/F)

In the second test, we apply a feature reduction family by family in order to differentiate between the Strings which are representative of a malware family.

We start by selecting the distinct Strings from each family in our dataset to build the vectors. For example, in Table 2, the malware family Agobot includes 340 files and has 1280 distinct Strings. For each file, we then generate a vector to indicate the presence of any of the 1280 distinct Strings.

We repeat the above for all the 11 families and use the vectors to generate the ARFF files to input in WEKA.

3.4 Feature Reduction

In this subsection, we proceed with the feature reduction phase of our experiment. We select the most important Strings which are rated based on their corresponding Information Gain values and retain a portion of the highest ranked Strings for the classification stage, as discussed in Section 4.

In WEKA, IG is a single attribute evaluator ([16], p.489) where IG evaluates an attribute based on its information gain. We use the ranker search method [11] for attribute selection, as it allows WEKA to rank attributes based on their individual IG evaluation.

At the end of the attribute selection process, we obtain a file for each family (and $mGroup_i$) with IG values for each of the Strings. The files are then ranked in terms of highest to lowest values. The next step is to determine the 'n' most important IG values with which to build the reduced feature set.

Since the families in our dataset have varying sizes, we had to ensure that we do not discriminate between families with large number of files against those with lesser files. Hence, we did some preliminary tests to determine the cut-off values. We start by finding the average number of Strings per family and use that number as a threshold. Finally, depending on the threshold value, we selected the Strings from the IG results. The final number of Strings selected from each family is shown in Table 3.

Table 3. Number of Strings Selected

Family	Number of Files in Family	Number Strings selected from IG results
Agobot	340	1067
Alureon	58	34
Bambo	70	45
Beovens	144	303
Boxed	366	534
Clagger	47	14
Emerleox	78	79
Looked	67	11
Robknot	119	140
Robzips	82	101
Clean	465	107

In order to obtain a reduced set of Strings for the M/C test, we compute the sum of the numbers for all the malware families (column three in Table 3) and divide the total by 3 (i.e. by the number of *mGroups*). We then use the resulting number to select the Strings ranked by IG. We refer to the reduced feature set for M/C as 'Information Gain Feature Reduction (IGFR – M/C)' and it consists of 1078 Strings.

The reduced set for the F/F test includes all the distinct Strings after the threshold value is applied. The final reduced list, referred to as IGFR-F/F includes 1266 Strings.

4 Experimental Work

4.1 The Classification Model

We summarize the steps in the classification model: the experimental data (both malware and cleanware files) are executed in a virtual machine; the execution behaviors are recorded in log files, which are collected at the end of the executable file run-time; the features are extracted from the log files and stored in the database table; 2 separate reduced feature sets are generated (IGFR-M/C and IGFR-F/F), which are then converted into vector form, representing the features' frequencies, to generate the input files to WEKA.

For the classification process, for the purpose of comparison, we use the same four WEKA classifiers as in [12] and apply 10-fold cross validation [8]. The classification model selects the malware executables for a particular family, *M*, and chooses the same number of files in the selected one at random from other families- referred to as *Other*, using a random function. The classifier then divides *M* into 10 groups of equal size as follows:

- If $10 \mid |M|$, then each group has size $\frac{|M|}{10}$.
- If $10 \nmid |M|$, write

$$|M| = 9 * B + r, \text{ where } 0 \leq r < 9,$$

and generate 9 groups of size *B* and place the remaining *r* executables in a 10^{th} group along with (*B*-*r*) randomly chosen executables from *M*. Once the 10 groups are formed, the classifier then generates their corresponding arff files to be used as input to WEKA. The same steps are repeated to divide the set, *Other,* into 10 groups and generate the arff files.

The classifier then takes 9 groups from *M* and *Other* to set up the training set and the remaining one group from *M* and *Other* is used as the testing set. The process is repeated for each of the 11 families. Moreover, the authors of [12] noted that when applying the meta-classifier AdaboostM1 on top of the base-classifiers, better accuracy results were obtained (pg. 5). Therefore, we only present classification results where Adaboost is applied.

4.2 Classification Results

In this section, we proceed with the data preparation for the classification stage. We commence by generating the input files in the format that is required by WEKA and then applying the algorithms to obtain the classification accuracies.

We use the IGFR-M/C and IGFR-F/F lists obtained from Table 3 to generate the feature vectors for the two tests: M/C and F/F, where each vector is made up of String frequencies within a file. The vectors are then used to construct the WEKA (.arff) files using our Java procedure 'WriteToArff'.

Table 4 compares the classification results obtained using the reduced feature sets for the IGFR tests with those from [12]. In this table, the weighted accuracies for the four meta-classifiers are indicated. Table 5 lists the false positives and false negatives for these same tests.

Table 4. Classification Accuracy with Adaboost (Weighted Average)

	Malware 2010 [12]		**Our method (IGFR)**	
	M/C	F/F	M/C	F/F
SMO	96.1	95.2	93.7	94.3
IB1	92.7	93.5	89	90
DT	97.1	93.9	94.5	94.5
RF	97.3	97.4	94.6	94.5
Number of features	7605 strings	7605 strings	1078 strings	1266 strings

Table 5. False Positives and False Negatives

	Malware 2010 [12]				**Our method (IGFR)**			
	M/C		F/F		M/C		F/F	
	FP	**FN**	**FP**	**FN**	**FP**	**FN**	**FP**	**FN**
SMO	0	0.03	0.017	0.084	0.0878	0.038	0.075	0.039
IB1	0.02	0.06	0.06	0.08	0.197	0.022	0.199	0.022
DT	0.01	0.04	0.02	0.11	0.075	0.035	0.074	0.036
RF	0.01	0.04	0.021	0.04	0.074	0.035	0.071	0.039

5 Analysis of Results

In this section, we analyze our results and compare them with the results from [12]. We also compare the tests on the basis of speed.

Figure 2 summarizes the weighted average accuracies by classifier for the four tests. Random Forest provides the best accuracy overall, while IB1 displays the largest spread (4.5%) across the four tests. For the M/C test, the smallest difference is 2.6% with DT; for the F/F test, the smallest difference is 0.6, also with DT. Note that in all cases but one, the test of [12] has better accuracy than IGFR; however, with DT, the IGFR test shows better results than those of [12] for family by family malware classification.

Turning to the false positives and false negatives identified in the tests, Figures 3 and 4 summarize the results of Table 5. In the false positive case, there are large discrepancies between the IGFR approach and that in [12] with the latter faring much better than the former; Figure 3 presents these discrepancies clearly. On the other hand, in the case of false negatives, Figure 4 indicates that the IGFR approach is superior to that in [12] in both tests. At this time, we are not able to explain why this would be the case.

Overall, then, the ability of the new tests to identify malware is close to that of the tests in [12] based on the identical data set, and in some cases surpasses that of [12]. Thus, a reduction in number of features used, if this is done well, can indeed produce good quality results. Recall that the IGFR-M/C test used 1078 strings and the IGFR-F/F test used 1266 strings. This contrasts with the tests of [12] which used 7605 strings for each test.

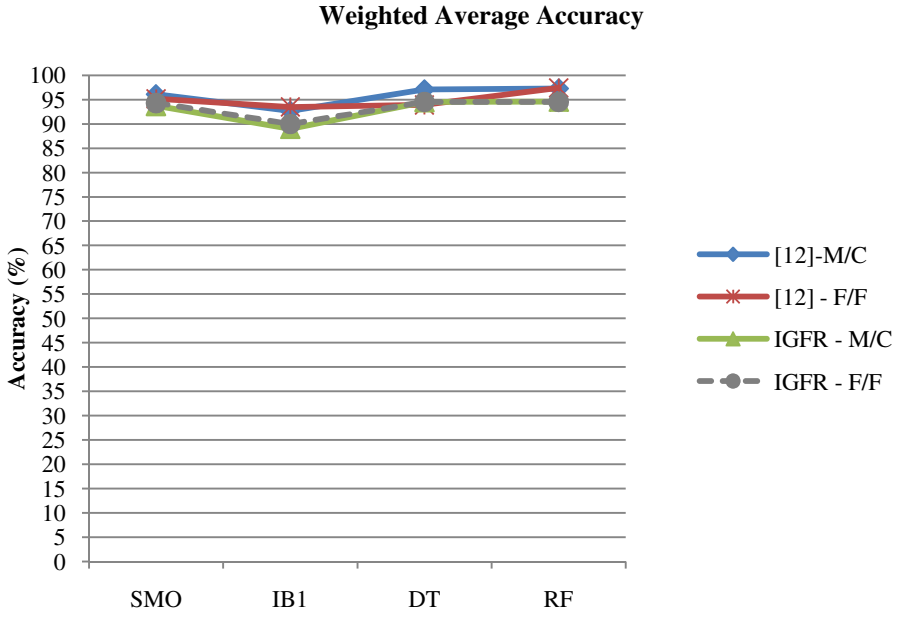

Fig. 2. Classification Accuracy

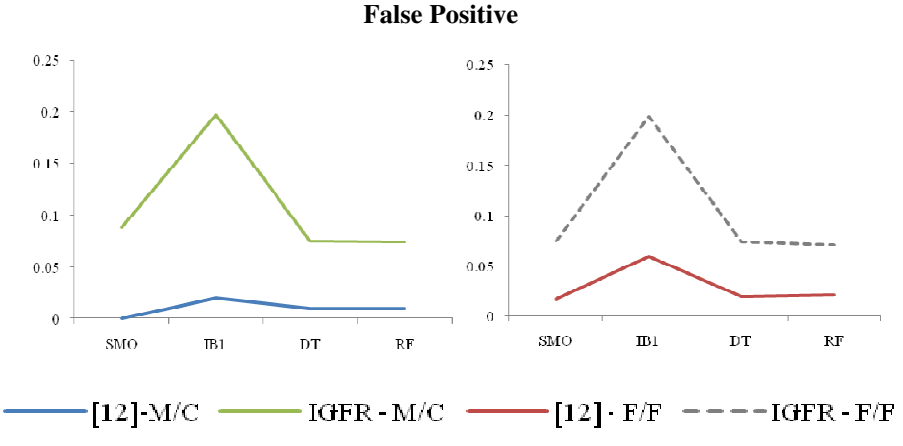

Fig. 3. Comparison of False Positive

We now turn to the question of speed with which the tests ran. Table 6 gives the total time in minutes for each of the four tests in question and Figure 5 presents a bar chart comparison of this data. The M/C test with the IGFR method ran in 65% of the time of the M/C test using the method of [12]. The IGFR-F/F test ran in 52% of the time of the F/F test using the method of [12]. Thus, both IGFR tests show a dramatic reduction in time needed to perform the tests while retaining good accuracy.

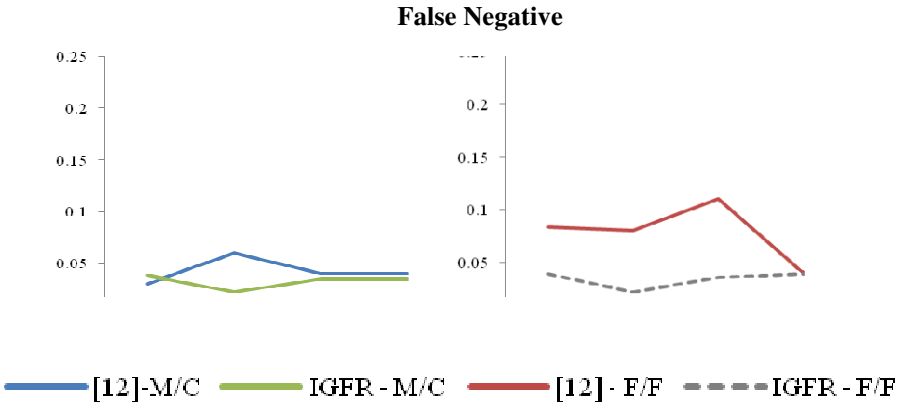

Fig. 4. Comparison of False Negative

Table 6. Total Classification Time (mins)

Experiment	Total Classification Time (mins)
[12] – M/C	402
[12] – F/F	560
IGFR – M/C	260
IGFR – F/F	293

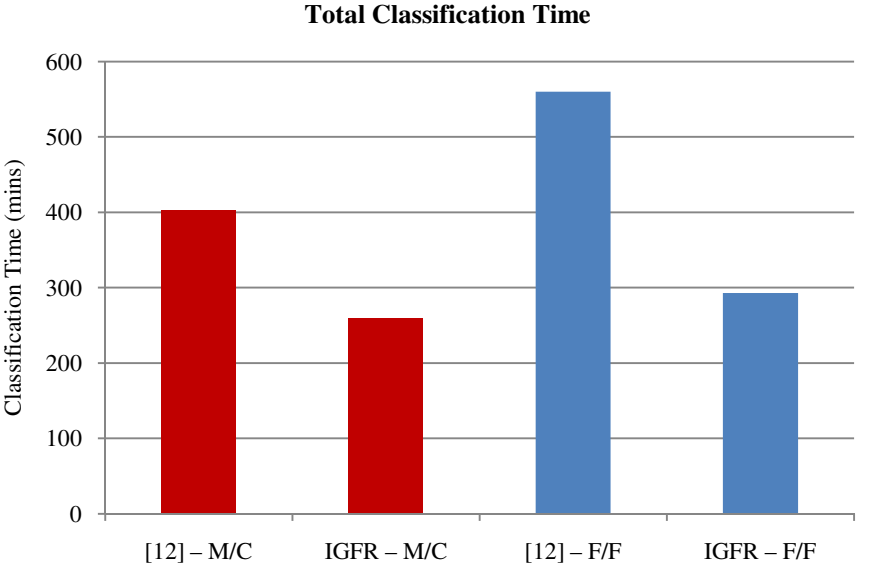

Fig. 5. Classification Time

6 Conclusions

We have taken advantage of a rare opportunity to re-use a data set from previous work ([12]) to compare running time and accuracy of that work against a new method proposed in this paper. Our aim was to significantly reduce the time needed to classify malware and to distinguish malware from cleanware. Table 6 and Figure 5 indicate that we were successful in achieving this as the time needed to perform each test was reduced by a factor of approximately 3/5. In addition, we were able to retain levels of accuracy of the results within 3% of the results of [12] on both tests.

The approach to saving time in our new method was to reduce the feature set by means of the statistical method Information Gain which is used to identify the most relevant features.

A feature of our work is the astonishing improvement we achieved in the false negative detection rate in classifying malware by family as presented in Figure 4. However, our approach did not improve on the false positive rate (Figure 3), and in future work, we shall investigate the possible reasons for this.

The small loss in accuracy and improved false negative rate along with significant improvement in speed of our tests indicate that feature reduction should be further pursued as a tool to prevent algorithms from becoming intractable due to the presence of too much data.

References

[1] Ahmed, F., Hameed, H., Shafiq, M.Z., Farooq, M.: Using spatio-temporal information in API calls with machine learning algorithms for malware detection. In: AISec 2009: Proceedings of the 2nd ACM Workshop on Security and Artificial Intelligence, pp. 55–62 (2009)

[2] Drozdz, K., Kwasnicka, H.: Feature Set Reduction by Evolutionary Selection and Construction. In: Jędrzejowicz, P., Nguyen, N.T., Howlet, R.J., Jain, L.C. (eds.) KES-AMSTA 2010. LNCS, vol. 6071, pp. 140–149. Springer, Heidelberg (2010)

[3] Henchiri, O., Japkowicz, N.: A feature selection and evaluation scheme for computer virus detection. In: Proceedings of the Sixth International Conference on Data Mining, ICDM 2006, pp. 891–895 (2006)

[4] Komashinskiy, D., Kotenko, I.: Malware detection by data mining techniques based on positionally dependent features. In: 2010 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), pp. 617–623 (2010)

[5] Kolter, J.Z., Maloof, M.A.: Learning to detect and classify malicious executables in the wild. Journal of Machine Learning Research 7, 2721–2744 (2006)

[6] Li, Y., Lu, B.: Feature selection based on loss-margin of nearest neighbor classification. Pattern Recognition 42(9), 1914–1921 (2009)

[7] Lu, Y., Din, S., Zheng, C., Gao, B.: Using multi-feature and classifier ensembles to improve malware detection. Journal of CCIT 39(2), 57–72 (2010)

[8] Masud, M.M., Khan, L., Thuraisingham, B.: Feature Based Techniques for Auto-Detection of Novel Email Worms. In: Zhou, Z.-H., Li, H., Yang, Q. (eds.) PAKDD 2007. LNCS (LNAI), vol. 4426, pp. 205–216. Springer, Heidelberg (2007)

[9] Mehdi, S.B., Tanwani, A.K., Farooq, M.: IMAD: in-execution malware analysis and detection. In: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation, GECCO 2009, pp. 1553–1560. ACM (2009)
[10] Moskovitch, R., Stopel, D., Feher, C., Nissim, N., Elovici, Y.: Unknown malcode detection via text categorization and the imbalance problem. In: 2008 IEEE International Conference on Intelligence and Security Informatics, pp. 156–161 (2008)
[11] Ranker Search Method, http://weka.sourceforge.net/doc.stable/weka/attributeSelection/Ranker.html
[12] Tian, R., Islam, R., Batten, L., Versteeg, S.: Differentiating malware from cleanware using behavioural analysis. In: Proceedings of the 5rd International Conference on Malicious and Unwanted Software: MALWARE 2010 (2010)
[13] Wang, T.-Y., Wu, C.-H., Hsieh, C.-C.: A virus prevention model based on static analysis and data mining methods. In: IEEE 8th International Conference on Computer and Information Technology Workshops, CIT Workshops 2008, pp. 288–293 (2008)
[14] Wang, T., Wu, C., Hsieh, C.: Detecting unknown malicious executables using portable executable headers. In: Fifth International Joint Conference on INC, IMS and IDC, NCM 2009, pp. 278–284. IEEE (2009)
[15] Waikato Environment for Knowledge Acquisition (WEKA): Data Mining Software in Java. University of Waikato, http://www.cs.waikato.ac.nz/ml/weka
[16] Witten, I., Frank, E., Hall, M.A.: Data mining: Practical machine learning tools and techniques, 3rd edn. Morgan Kaufmann, Burlington (2011)
[17] Ye, Y., Li, T., Jiang, Q., Wang, Y.: CIMDS: Adapting Postprocessing Techniques of Associative Classification for Malware Detection. IEEE Transactions on Systems, Man, and Cybernetics 40(3), 298–307 (2010)