**Evaluation of supervised and unsupervised machine learning classifiers for Mac OS malware detection**

Dilip Sahoo, Yash Dhawan

[dsahoo@uoguelph.ca](mailto:dsahoo@uoguelph.ca), ydhawan@uoguelph.ca

School of Computer Science, University of Guelph, Ontario, Canada

Abstract

Mac operating system is based on UNIX based platform and some consider it to be a more resilient operating system compared to the Windows platform. However, the number of attacks for Mac OS has increased exponentially over recent years and new attacks are arising daily which is capable of bypassing the Mac inbuilt security mechanism. Various supervised and unsupervised machine learning classifiers can be used to detect malware samples by comparing their behavior such as the system calls with benign apps. In this paper, we have evaluated five different supervised and unsupervised classifiers to distinguish between the good ware and malware samples of the Mac OS platform. The experiment was conducted using two different approaches: using the original dataset and then a synthetic balanced dataset. We used Synthetic Minority Over-sampling Technique (SMOTE) for the upsampling of minority class and train the classifiers with a balanced dataset. The experiment results show that the balanced dataset reduces bias towards the majority class and increases the machine learning classifiers' accuracy. Using this approach, we successfully achieved higher accuracy for five machine learning algorithms with a low false-positive rate.

Keywords:Mac OS, Malware detection, Machine Learning, Mac OS malware

# Introduction

The number of attacks targeting MacOS has considerably risen in the past couple of years[1]. It is estimated that reported attacks have exceeded 4 million as of 2018 and another 1.8 million attacks have been reported during the first half of 2019[2]. The first Mac malware was reported in 2004 with Renepo script worm which disabled Mac OSX security and installed malicious toolkit[3]. Adware and Potential Unwanted Program (PUP) resulted in a serious threat for Mac users over the past couple of years as it resulted in security vulnerability making it more likely to get infected by malware[4]. Though the Mac platform is considered safer than Windows it is still prone to several phishing attacks, java-based exploit, the man in the middle attacks, and should not be considered as a bulletproof operating system[5].

Protecting IT resources and computer hardware against malware threats has become vital for corporations and individuals. Most antivirus (AV) software use a signature-based technique to detect the threats. A signature of the known malware like spyware, viruses, trojans, worms is stored in a database and if an attack occurs by them in the future then they can be detected against their stored signatures. However, there are many drawbacks to this approach of malware detection. Firstly, the signature-based approach is ineffective against the new malware that is not known previously. Secondly, the metamorphic malware (a variant of known malware) can bypass the antivirus by changing its signature[6]. Significant improvements have been done to make the AVs more effective using more sophisticated analysis techniques in recent years. However, there is still the problem of delay between the detection of new malware and updating the signature databases to counter it. This delay can cause significant damage to corporations[7].

In the last decade, more sophisticated methods are being used by researchers for metamorphic malware detection like dynamic and heuristic analysis. In dynamic analysis, the behavior of the malware program is observed at runtime in an isolated sandbox environment like a virtual box. During this process, specific behaviors of a program like system calls, registry updates, network traffic usage, etc are monitored and used to classify whether the program is a benign application or malware. However, this method can be time-consuming, and sometimes evasive methods used by the malware can detect the analysis environment and stops the malicious code execution or delay the execution[8]. Another main disadvantage is that dynamic analysis cannot be used in realtime scenarios.

In contrast, the heuristic approach uses Machine learning(ML) to learn the malicious program behaviors and can classify them as malware. They are easy to implement and can also effectively detect metamorphic malware. The disadvantage of the heuristic approach is higher false positives i.e the benign programs incorrectly classified as malware. To overcome this issue, the machine learning classifiers need to be trained with datasets with sufficient features and have a balanced ratio between the majority and minority class. A dataset with a large number of features can decrease the false positives but it can increase the overall computation and processing time and hence it is important to analyze and reduce the dataset dimension. There is the number of automated tools and application available which provide tons of features to neutralize a threat before it can compromise a system[9]. For an effective antimalware solution, many companies have adopted the machine learning approach[10]. ML models are trained to distinguish malicious and benign apps using supervised and unsupervised classifiers[11]. In the supervised learning method, a labeled dataset of malware and good ware is used for the training of the ML classifier. After sufficient training with labeled data, the ML model is used to classify the unseen samples. whereas, in the unsupervised learning method, the classification is done on observed similarities or differences[12].

In this paper, we employed a heuristic approach using the same raw Mac OSX dataset used by H.H Pajouh et al. (2017)[13], and a comprehensive study was done to evaluate different machine learning classifiers for the detection of MacOSX based malware samples. Below are the measure research work done as part of the experiment.

1. The experiment applied a (Term Frequency — Inverse Data Frequency) TF-ID based text processing was done to extract 654 new features based on the calling of application libraries.
2. A SMOTE data set was developed with balanced distributions of benign and malware samples to reduce bias in favor of any particular class. Each of the classifiers was evaluated using the SMOTE data and a thorough comparison was made against the original dataset and result.
3. Five different machine learning algorithms ( 4 Supervised and 1 Unsupervised machine learning technique) namely Logistic regression, Random Forest, Decision Tree, Naïve Bayes, and K-nearest neighbor were evaluated and analyzed from different aspects like accuracy, False Positive rate, processing time.

We used the commonly used matrices for the performance evaluation of the machine learning classifiers used in the experiment i.e True Positive Rate(TPR), False Positive Rate(FPR), Precision, Recall, F-measure, Reciever Operating Characteristics(ROC), and Area Under the ROC Curve (AUC). Detailed descriptions of the evaluation measures are described in sections 4.1 and 4.4.

.

Section 2 of this paper contains related work from recent years. Section 3 details the methodology used in this work, section 4 discuss the results observed in the experiment, and in section 5 we provide the conclusion of our work and suggest future work. Acknowledgment and References for our work are provided at the very end of the paper.

# Related Work

The threat landscape for MacOS is changing drastically as the amount of malicious software is growing[2]. The velocity, volume, and complexity of malware are posing numerous challenges for the antivirus companies[1]. Various supervised and unsupervised machine learning techniques are proving to be efficient in the detection of malware.

Ransomware attacks rose drastically ever since the introduction of cryptocurrencies through which attackers were able to receive ransom anonymously. The majority of ransomware families have different versions and features which makes their detection and analysis sophisticated. To resolve this T Dargahi[11] research provides the first scientific taxonomy of ransomware features, aligned with the Lockheed Martin Cyber Kill Chain (CKC) model. A comprehensive taxonomy would assist researchers in assessing the vulnerability and attack vectors towards intended victims. Sajad .H[14] proposed DRTHIS: Deep ransomware threat hunting and intelligence system which can detect ransomware activities utilizing Long Short-Term Memory (LSTM) and Convolutional Neural Network (CNN), two deep learning techniques. In the classification of ransomware instances, DRTHIS achieved an F-measure of 99.6% with a true positive rate of 97.2%. DRTHIS accurately predicted the ransomware instances based on sequences of action performed by good ware and ransomware samples to correctly classify ransomware samples.

Fattori et al[15]. developed an AccessMiner behavioral malware protection system that provides a high level of OS protection ( around 90 % with zero false positives). It generally detects malicious samples in real-time by monitoring interactions between applications and the Windows Operating system. Novel Active Learning (AL) framework introduced by Nissim[16] assisted antivirus vendors in determining more malware samples than the existing AL method. It provided an accuracy of 97 % as well as provided an increased efficiency to detect novel Windows malware. To reduce the chance of Malware evasion Mangialardo and Duarte[17] proposed the unification of Static and Dynamic analysis using C5.0 and Random Forests (RF) algorithms with an accuracy of 93% for detecting Linux malware.

Since Windows OS is used extensively, a prominent amount of research work has been conducted as compared to OSX malware detection[18]. H.H Pajouh proposed an OSX code inspection technique using Synthetic Minority Over-sampling Technique (SMOTE) to improve malicious sample size in the dataset which helped to achieve a higher malware detection accuracy of 96 % and achieve a lower false alarm rate[13]. Some other relevant research includes researcher Pham Duy Phuc used MacOS a malware analysis framework called Mac-A-Mal to automatically capture malware behavior at user and kernel levels. Mac-A-Mal framework led to the discovery of 71 unknown Adware, 2 keyloggers, and 1 trojan involved in the APT32 OceanLotus. It also provided a Heatmap correlation matrix to analyze the correlation of different malware datasets. The model supported static and dynamic analysis to provide a rich set of Mac malware variants that machine learning classifiers can implement[19].E. Walkup[20] implemented static executable analysis for the detection of Mac malware using different supervised classification techniques. Information gain was utilized in the dataset to select prominent features to detect OS X malware. Our machine learning classifiers aim to solve the gap that was observed in the above papers.

# Methodology

This section describes the experiment workflow as shown in Figure 1. First, we obtained the OSX Malware Detection dataset from the *cybersciencelab.org* [21]website which had 450 benign samples and 152 malware samples. The raw data was then processed to remove the anomalies. Second, different feature selection and feature extraction techniques were used on the processed data, and in the final stage different machine learning classifiers were used to create a detection model. Each step is described in detail in sections 3.1, 3.2, and 3.3 respectively.

Two different experiments were conducted following the above steps. The first experiment was conducted using the actual original data that was used to train and test different ML classifiers. In the second experiment, SMOTE data was used which was developed using the oversampling technique. All the experiments were conducted using Python 2.7.1, Jupyter notebook server version 6.0.1 in a Windows 7 virtual machine with intel i7(2.20GHz) processor.

Figure 1 Experiment workflow

## Data Preprocessing

In this step, we analyzed the MacOSX Malware Detection raw data and found multiple anomalies were found. Each of these anomalies was removed after a thorough inspection. Some columns were having data in both integers and in hex format, which were converted to decimal format so that the features can be of integer type. The null values and bad data were replaced with the mean value of the respective columns. Later this data was converted to CSV (comma-separated values) format and used in the next phase for the feature selection and feature extraction process.

## Feature Selection and Extraction

After the data preprocessing phase, the most relevant features were selected using two different feature selection techniques. We used *ExtraTreeClassifier* [22]form the scikit-learn library for statistical analysis of the feature importance of the dataset. This analysis gave an idea about which features are more relevant as shown in Figure 2.We observed that features like LoadDYLIB, bind\_size, rebase\_size, and ncmds (Descriptions are given in Table 2) have more importance relative to other features to determine the final output and classification.

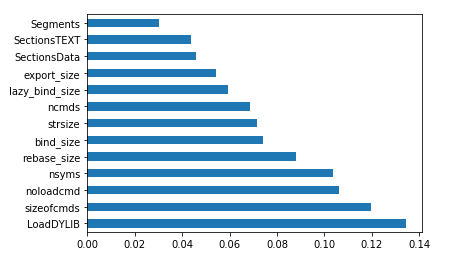


Figure. 2 Feature scores obtained using ExtraTreeClassifier

Principal Component Analysis was used to understand how many dimensions of the data maximize the variance of the dataset. This gave us the idea that the first eight dimensions explain approx 90% of the variance as shown in Table 1.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| D1 | D2 | D3 | D4 | D5 | D6 | D7 | D8 | D9 | D10 | D11 | D12 | D13 |
| 0.345 | 0.507 | 0.606 | 0.682 | 0.753 | 0.813 | 0.862 | 0.906 | 0.940 | 0.966 | 0.984 | 0.993 | 1.000 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |

Table 1 Variance by Dimension

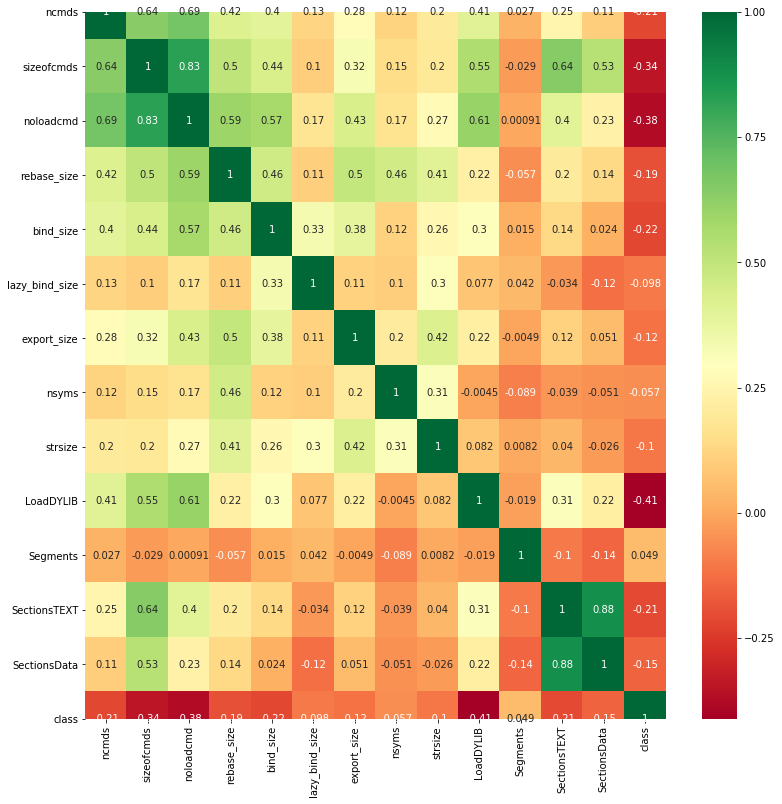


Figure. 3 Heatmap Correlation Matrix for the features

Heatmap Correlation Matrix, shown in Figure 3 was used to understand the correlation between different features in the dataset. We observed that the features like ncmds, sizeofcmds, noloadcmd, rebase\_size, and bind\_size have a greater correlation with the output compared with other features in the dataset. Segments, SectionsTEXT, and SectionsData have the least correlation with the output variable.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Feature | name | Type | Description | Data Type |
| 1 | ncmds | Integer | No. of commands for every sample Integer | Integer |
| 2 | sizeofcmds | Integer | Command size for every sample | Integer |
| 3 | noloadcmd | Integer | Number of commands for every loaded sample during execution | Integer |
| 4 | rebase\_size | Integer | Describe the size of the rebase info Integer | Integer |
| 5 | bind\_size | Integer | Describing the size of the info to be bound during execution | Integer |
| 6 | lazy\_bind\_size | Integer | States the size of the info to be bound during execution | Integer |
| 7 | export\_size | Integer | States the size of lazy binding info Integer | Integer |
| 8 | nsyms | Integer | States the no of symbol table entries Integer | Integer |
| 9 | strsize | Integer | States size of string table in bytes | Integer |
| 10 | LoadDYLIB | Integer | States no of DYLIB called and load for execution of malware | Integer |
| 11 | Segments | Integer | Number of total segments which consist of every sample | Integer |
| 12 | SectionsTEXT | Integer | No of text segments consisting of every sample | Integer |
| 13 | SectionsData | Integer | No of data segments consisting of every sample | Integer |
| 14 | DYLIBnames | String | Define names of loaded DYLIB | String |

Table 2 Features description of Mac OS dataset. Description are adopted from the paper on OSX malware detection [13]

### Feature extraction

The column DYLIBnames in the data set has the information regarding system libraries called by applications. Names of different libraries were stored in a comma-separated string format. We did the text processing of the column separately and extracted several features using *TfidfVectorizer* from scikit-learn (https://scikit-learn.org). During this process, we first used ‘comma’ as the token separator to split each library's names from the strings in each sample and were then converted to new feature columns. The Term Frequency — Inverse Data Frequency (TFIDF) [23] values of these libraries were calculated and assigned to the new feature columns for each sample. Finally, 654 new features were extracted from the DYLIBnames column and were also used for training and testing the ML classifiers.

### SMOTE dataset development

### 

Balancing the data is crucial because an imbalanced dataset can cause biased results in favor of the majority class[24]. Our original dataset contained 450 samples from the benign class and 152 from malware class which was in the ratio of approx 3 to 1. Hence, a SMOTE dataset was developed with an oversampling technique to balance the minority class. Table 3 illustrates the data sample details for original and SMOTE datasets.

|  |  |  |  |
| --- | --- | --- | --- |
| Dataset | Benign | Malicious | Total |
| Origional | 460 | 152 | 612 |
| SMOTE | 460 | 460 | 920 |

Table 3 Original dataset and the SMOTE dataset sample distributions

## Machine learning classifier phase

Five different machine learning classifier namely *LogisticRegression, Random Forest, KNN, decision tree,* and *Naïve Bayes* were used where KNN is an unsupervised classifier, and rest are supervised classifiers. The ML model is trained and tested in two stages. In the first stage, the original dataset was used along with the 654 newly extracted features, and in the second stage, a SMOTE enhanced dataset was used to train and test all the classifiers.

# Experiment and Results

This section describes the results obtained from our experiments and the performance evaluation of different ML classifiers used during the experiment. The results were obtained using a 10 fold cross-validation technique. The assessments were done in two phases. First, the actual results from different ML classifiers were obtained with the original dataset and in the second phase, the SMOTE dataset was used and results were recorded. Later, more analysis was done by making comparisons of important performance metrics like Accuracy, ROC curves, False Positive Rate, and processing time. Finally, the results were compared with another similar paper.

## Evaluation Measures

Output results of the ML classifiers can be summarised as a confusion matrix where the diagonal elements are True Positives(TP), True Negatives(TN), and the off-diagonal elements as False Positives(FP) and False Negatives(FN) respectively.

**True positive (TP):** The normal observation is predicted as normal

**True Negative(TN):** Anomalous observation is predicated as anomalous

**False Positive(FP):** Anomalous observation is predicted as normal

**False Negative(FN):** Normal observation is predicated as anomalous

True Positive Rate (TPR) =

False Positive Rate (FPR) =

Precision =

Recall = TPR =

F-measure =

Accuracy =

Here TPR, also known as Recall is the value of predicted malware classified correctly and FPR is the value of normal data incorrectly predicted as malware. Precision is also known as the positive predicted value, returns the rate of relevant results. F-measure provides value that estimates the entire system performance by combining precision and recall into a single number. Accuracy denotes how accurately an ML classifier can classify the binary classes i.e ‘good ware ’ and ‘malware’

## Evaluation Of ML classifiers

We have used different performance metrics namely: Precision, Recall, F-measure, and Accuracy to evaluate our ML models. Table 4 shows a summary of the overall results obtained from the two experiments i.e using normal dataset and the SMOTE dataset. In the first experiment, Logistic regression achieved the highest detection rate of 0.94 in terms of testing accuracy. Random Forest and then KNN came next and achieved an accuracy of 0.93 and 0.88 respectively.

In the second experiment, an overall increment in all parameters of the detection result was observed for the five classifiers when the SMOTE dataset was used. Especially, for Decision Tree and Naïve Bayes, a significant increment of approx 10% was observed in the accuracy. In the second experiment also Logistic Regression remained on the top with a detection accuracy of 0.96 followed by Random Forest with 0.95, Decision Tree with 0.94, Naïve Bayes with 0.93, and KNN with 0.90. Surprisingly, Decision Tree and Naïve Bayes outperformed KNN in terms of all the parameters i.e precision, Recall, F-measure, and testing accuracy during this experiment.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Experiment | No. | Classifier | Precision | Recall | F-measure | Accuracy |
| I. Original dataset | 1 | Logistic Regression | 0.94 | 0.94 | 0.94 | 0.94 |
| 2 | Random Forest | 0.94 | 0.93 | 0.93 | 0.93 |
| 3 | KNN | 0.89 | 0.88 | 0.88 | 0.88 |
| 4 | Decision Tree | 0.87 | 0.87 | 0.87 | 0.87 |
| 5 | Naïve Bayes | 0.86 | 0.83 | 0.84 | 0.83 |
| II. SMOTE dataset | 1 | Logistic Regression | 0.96 | 0.96 | 0.96 | 0.96 |
| 2 | Random Forest | 0.95 | 0.95 | 0.95 | 0.95 |
| 3 | KNN | 0.91 | 0.9 | 0.89 | 0.90 |
| 4 | Decision Tree | 0.94 | 0.94 | 0.93 | 0.94 |
| 5 | Naïve Bayes | 0.94 | 0.93 | 0.93 | 0.93 |

Table 4 Result summary comparing evaluation result from the Original dataset and the SMOTE dataset

## False-Positive Rate comparison

Tables 5 shows the count of predicted TP, FN, FP, TN while testing each of the ML classifiers. Then, FPR was calculated and compared for both the experiments using original data and SMOTE data respectively. A significant reduction in FPR was recorded for the SMOTE dataset compared to the original dataset.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Experiment | No. | Classifier | TP | FN | FP | TN | FPR |
| I. Original data | 1 | Logistic Regression | 91 | 3 | 4 | 25 | 0.14 |
| 2 | Random Forest | 94 | 0 | 8 | 21 | 0.28 |
| 3 | KNN | 83 | 11 | 4 | 25 | 0.14 |
| 4 | Decision Tree | 85 | 9 | 7 | 22 | 0.24 |
| 5 | Naïve Bayes | 78 | 16 | 5 | 24 | 0.17 |
| II. SMOTE data | 1 | Logistic Regression | 129 | 8 | 3 | 137 | 0.02 |
| 2 | Random Forest | 128 | 9 | 5 | 135 | 0.04 |
| 3 | KNN | 109 | 28 | 1 | 139 | 0.01 |
| 4 | Decision Tree | 126 | 11 | 7 | 133 | 0.05 |
| 5 | Naïve Bayes | 117 | 20 | 0 | 140 | 0.00 |

Table5 TP, TN, FP, TN count, and FPR value for ML classifiers tested with the original dataset and SMOTE dataset

Figure 4 shows the false-positive rate (FPR) comparison between the two experiments using the above mentioned different datasets for each classifier. It was observed that with the original dataset Random Forest had the highest false-positive rate i.e 27.59%, followed by Decision Tree at 24.14%. For both KNN and Logistic Regression, the FPR was relatively low at 13.79%. After running the ML classifiers on the SMOTE dataset, there was a significant decrease in the FPR value for all the classifiers. Naive Bayes achieved a noteworthy 0% FPR rate with SMOTE dataset. The difference in false alarm for a small amount of data won’t be significant, but practically the traffic over any common network is huge enough that the difference can be observed.

Figure 4 False alarm (FPR) comparison of the ML classifiers for Original dataset and SMOTE dataset

## ROC Curve

Figure 5 and Figure 6 shows the comparison of the Receiver Operating Characteristic (ROC) Curve for the ML Classifiers. ROC curve[25] is one of the methods of measuring the performance of a classification model. In this curve, the True Positive Rate (TPR) is plotted against False Positive Rate (FPR) for the probabilities of the classifier predictions. Then, the area under the plot is calculated. More the area under the curve, better is the model at distinguishing between classes.

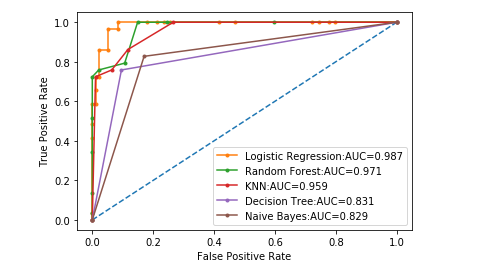
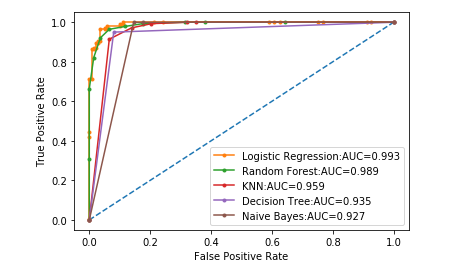
 

Figure 6 ROC curve for the ML classifiers for SMOTE dataset

Figure 5 ROC curve for the ML classifiers for Original dataset dataset

In Figure 5 and Figure 6, the graphs illustrate that the Logistic Regression has the highest AUC of 0.987 and 0.993 for the original dataset and SMOTE dataset respectively, which signifies an outstanding prediction score. AUC for RF is was recorded a little lower than Logistic Regression at 0.971 and 0.989 for the mentioned datasets denoting very good prediction as well. Finally, followed by KNN, Decision Tree, and Naïve Bayes for the same datasets. It was observed that there was an increment in AUC value for each of the ML classifiers when the SMOTE dataset was used.

## Performance Evaluation

In this section performance evaluation of each classifier in terms of processing time is discussed. Figure 7 shows the execution time comparison between the two experiments using the original data set and SMOTE dataset. First, with the original dataset, Logistic Regression and Naïve Bayes performed better than the rest of the algorithms, where both took 170ms. In experiment-I, Random Forest took the highest execution time of 460ms. Later, with the SMOTE dataset again Logistic Regression and Naïve Bayes performed better than other classifiers and a slight decrease in execution time was also noted for them. For KNN an increase in the execution time was recorded when SMOTE dataset was used, where it took 470 ms compared to 350 ms in the first experiment. Other than KNN the execution time either reduced or remained almost the same in the second experiment with the SMOTE dataset.

Figure 7 performance comparison based on the execution time for the ML classifiers

Figure 10 performance comparison based on the execution time for the ML classifiers

## Result Comparison

Authors in [13] researched macOS malware detection using the same dataset. They used three different SMOTE enhanced i.e 2x\_SMOTE, 3x\_SMOTE, and 5x\_SMOTE dataset for their experiment. However, the benign and malware sample was always in the ratio of approx 3:1 in the datasets like it was in the original dataset. The sample weight of libraries was used with an occurrence probability value for the new feature creation. Decision achieved the highest accuracy of 96% and 4% false alarm with the 5x\_SMOTE dataset. weighted RBF-SVM achieved 91% accuracy with a 3.9% false alarm using the original dataset.

In this paper, we used SMOTE data with a balanced distribution for both benign and malware samples in the ratio of 1:1. The library calls frequency was calculated using TF-IDF text processing method and was used for the new feature creation. In our experiment, the highest accuracy was obtained from Logistic regression which was 96% with a false alarm of 2.14% using SMOTE data. Using original data, Logistic regression achieved the highest accuracy of 94% and a false alarm of 13.79%. Overall, all the classifiers used in this paper show better detection accuracy and a lower false-positive rate.

# Conclusion and future work

In this paper, we used TF-ID based text processing to extract new features, and using the Synthetic Minority Over-sampling Technique (SMOTE) we were able to balance our dataset and achieve better results compared to our original dataset. We tested our dataset with 5 machine learning classifiers ( 4 Supervised and 1 Unsupervised algorithm) and achieved a promising accuracy of 96 % for Logistic regression and 95 % for Random forest. The experiment also provided us with a lower false-positive rate as evident in figure 8.

Future work would include malware detection and analysis to be further improved by acquiring more data samples to create a better detection system. Additional research can be done on a larger dataset to predict if the same results are obtained or not. To acquire predictive performance ensemble machine learning technique works well-using bagging and boosting techniques using several base models. With Ensemble machine learning we can reduce variance, noise, and bias as well as increases the accuracy of the model[26]. A deep learning approach can also be used on a large malware dataset to detect complex malware and outperform traditional machine learning algorithms[27]. This paper can be treated as a case study for researchers working in the area of intelligent MacOS malware detection systems on enterprise and cloud platforms.

**References:**

[1] “Mac Malware Getting Serious - Security No Longer Optional,” *Macworld*, Feb. 05, 2018. https://www.macworld.com/article/3253252/mac-malware-getting-serious-security-no-longer-optional.html (accessed Oct. 24, 2019).

[2] “Threats to macOS users.” https://securelist.com/threats-to-macos-users/93116/ (accessed Oct. 22, 2019).

[3] “Mac Malware Facts.” https://www.eset.com/int/mac-malware-facts/ (accessed May 10, 2020).

[4] T. Reed, “Mac security facts and fallacies,” *Malwarebytes Labs*, Mar. 08, 2017. https://blog.malwarebytes.com/101/2017/03/mac-security-facts-and-fallacies/ (accessed May 10, 2020).

[5] “Warning as Mac malware exploits climb 270% | Computerworld.” https://www.computerworld.com/article/3262225/warning-as-mac-malware-exploits-climb-270.html (accessed May 10, 2020).

[6] Y. Ye, T. Li, Q. Jiang, and Y. Wang, “CIMDS: adapting postprocessing techniques of associative classification for malware detection,” *IEEE Trans. Syst. Man Cybern. Part C Appl. Rev.*, vol. 40, no. 3, pp. 298–307, May 2010, doi: 10.1109/TSMCC.2009.2037978.

[7] S. Homayoun, A. Dehghantanha, M. Ahmadzadeh, S. Hashemi, and R. Khayami, “Know Abnormal, Find Evil: Frequent Pattern Mining for Ransomware Threat Hunting and Intelligence,” *IEEE Trans. Emerg. Top. Comput.*, vol. PP, pp. 1–1, Sep. 2017, doi: 10.1109/TETC.2017.2756908.

[8] S. K. Sahay and A. Sharma, “Grouping the Executables to Detect Malwares with High Accuracy,” *Procedia Comput. Sci.*, vol. 78, no. C, pp. 667–674, Mar. 2016, doi: 10.1016/j.procs.2016.02.115.

[9] P. N. Bahrami, A. Dehghantanha, T. Dargahi, R. M. Parizi, K.-K. R. Choo, and H. H. S. Javadi, “Cyber Kill Chain-Based Taxonomy of Advanced Persistent Threat Actors: Analogy of Tactics, Techniques, and Procedures,” *J. Inf. Process. Syst.*, vol. 15, no. 4, pp. 865–889, Aug. 2019.

[10] “How machine learning works | Kaspersky official blog.” https://www.kaspersky.com/blog/machine-learning-explained/13487/ (accessed May 10, 2020).

[11] T. Dargahi, A. Dehghantanha, P. N. Bahrami, M. Conti, G. Bianchi, and L. Benedetto, “A Cyber-Kill-Chain based taxonomy of crypto-ransomware features,” *J. Comput. Virol. Hacking Tech.*, vol. 15, no. 4, pp. 277–305, Dec. 2019, doi: 10.1007/s11416-019-00338-7.

[12] E. McNulty, “What’s The Difference Between Supervised and Unsupervised Learning?,” *Dataconomy*, Jan. 08, 2015. https://dataconomy.com/2015/01/whats-the-difference-between-supervised-and-unsupervised-learning/ (accessed May 10, 2020).

[13] H. H. Pajouh, A. Dehghantanha, R. Khayami, and K.-K. R. Choo, “Intelligent OS X malware threat detection with code inspection,” *J. Comput. Virol. Hacking Tech.*, vol. 14, no. 3, pp. 213–223, Aug. 2018, doi: 10.1007/s11416-017-0307-5.

[14] S. Homayoun *et al.*, “DRTHIS: Deep ransomware threat hunting and intelligence system at the fog layer,” *Future Gener. Comput. Syst.*, vol. 90, Jul. 2018, doi: 10.1016/j.future.2018.07.045.

[15] A. Fattori, A. Lanzi, D. Balzarotti, and E. Kirda, “Hypervisor-based malware protection with AccessMiner,” *Comput. Secur.*, vol. 52, pp. 33–50, Jul. 2015, doi: 10.1016/j.cose.2015.03.007.

[16] N. Nissim, R. Moskovitch, L. Rokach, and Y. Elovici, “Novel active learning methods for enhanced PC malware detection in windows OS,” *Expert Syst. Appl.*, vol. 41, no. 13, pp. 5843–5857, Oct. 2014, doi: 10.1016/j.eswa.2014.02.053.

[17] R. J. Mangialardo and J. C. Duarte, “Integrating Static and Dynamic Malware Analysis Using Machine Learning,” *IEEE Lat. Am. Trans.*, vol. 13, no. 9, pp. 3080–3087, Sep. 2015, doi: 10.1109/TLA.2015.7350062.

[18] “Operating System Guide: Windows vs Mac (vs Linux).” https://www.logicalincrements.com/articles/build-pc-windows-apple-mac-linux-operating-system-os (accessed May 10, 2020).

[19] D.-P. Pham, D.-L. Vu, and F. Massacci, “Mac-A-Mal: macOS malware analysis framework resistant to anti evasion techniques,” *J. Comput. Virol. Hacking Tech.*, vol. 15, no. 4, pp. 249–257, Dec. 2019, doi: 10.1007/s11416-019-00335-w.

[20] E. Walkup, “Mac Malware Detection via Static File Structure Analysis,” *Univ. Stanf.*, [Online]. Available: http://cs229.stanford.edu/proj2014/Elizabeth%20Walkup,%20MacMalware.pdf.

[21] “CSL-Home - Cyber Science Lab.” https://cybersciencelab.org/ (accessed May 10, 2020).

[22] “3.2.4.3.3. sklearn.ensemble.ExtraTreesClassifier — scikit-learn 0.22.2 documentation.” https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesClassifier.html (accessed May 10, 2020).

[23] “How to process textual data using TF-IDF in Python,” *freeCodeCamp.org*, Jun. 06, 2018. https://www.freecodecamp.org/news/how-to-process-textual-data-using-tf-idf-in-python-cd2bbc0a94a3/ (accessed May 10, 2020).

[24] K. Mahendru, “How to Deal with Imbalanced Data using SMOTE,” *Medium*, Jun. 26, 2019. https://medium.com/analytics-vidhya/balance-your-data-using-smote-98e4d79fcddb (accessed May 10, 2020).

[25] “Machine Learning Classifier evaluation using ROC and CAP Curves.” https://towardsdatascience.com/machine-learning-classifier-evaluation-using-roc-and-cap-curves-7db60fe6b716 (accessed May 10, 2020).

[26] R. R. F. DeFilippi, “Boosting, Bagging, and Stacking — Ensemble Methods with sklearn and mlens,” *Medium*, Aug. 04, 2018. https://medium.com/@rrfd/boosting-bagging-and-stacking-ensemble-methods-with-sklearn-and-mlens-a455c0c982de (accessed May 10, 2020).

[27] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, and S. Venkatraman, “Robust Intelligent Malware Detection Using Deep Learning,” *IEEE Access*, vol. 7, pp. 46717–46738, 2019, doi: 10.1109/ACCESS.2019.2906934.