Malware Analysis Project report.

Name: **Oluwatoyin Sanni**
Matricola: **1871835**
Department: **Artificial Intelligence and Robotics**
November 18th, 2018

This report is accompanied with two major files:

- A python script named *"data_extraction.py"* which is the script used to extract the datasets from the original downloaded data. This part of the code was done in collaboration.
- A jupyter notebook which contains my exploratory and model code, this was solely written by me.

I. Definition

Project Overview

According to wikipedia, **Malware analysis** is the study or process of determining the functionality, origin and potential impact of a given malware sample such as a virus, worm, trojan horse, rootkit, or backdoor. Malware or malicious software is any computer software intended to harm the host operating system or to steal sensitive data from users, organizations or companies.

Malware may include software that gathers user information without permission. It's no doubt that the Academic and industry malware researchers may perform malware analysis simply to understand how malware behaves and the latest techniques used in its construction.

In this project, I explored malware analysis using the Debrin dataset obtained from Android phone users. Past research have been done in the past using the same Debrin dataset, an example is this paper which adopts the linear Support Vector Machines for same classification task, their model was able to reliably detect all families with an average accuracy of 93% at a false-positive rate of 1%. In my work, I will be exploring other models to arrive at a better score, I also employed other metrics other than accuracy.

Problem statement

The dataset for this problem contains 5,560 applications from 179 different malware families. The samples have been collected in the period of August 2010 to October 2012 and were made available to by the MobileSandbox project through the Technische Universität Braunschweig, Germany.

This project aims to propose a befitting function to classify android malicious software to protect sensitive data against malicious threats using data mining and machine learning classification techniques. In this work, I employed a robust and efficient approach for malware classification by analyzing the data while exploring techniques to combat the imbalanced datasets. Simply put, a binary classification to determine if an application is malware or not.

Metrics

As seen in other research mentioned in my overview, accuracy seems to be the most popular benchmark metric to validate machine learning models. However, I explored other metrics; f1 score, area under curve, and lastly Precision-Recall Curve which is the eventual benchmark metric.

I find precision-recall curve much more convenient in this case as our problems relies on the "positive" class being more interesting than the negative class.

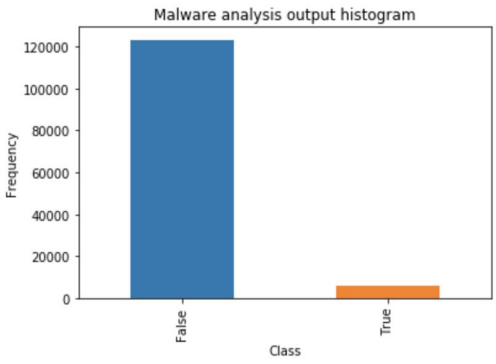


Fig. 1

Also, for imbalanced datasets like ours in Fig. 1, accuracy score will not be effective enough because it's just the number of correct predictions made divided by the total number of predictions made, multiplied by 100 to turn it into a percentage.

For problems like our imbalanced datasets, this following measures are required to evaluate a classifier.

- **Precision**: Precision is the number of True Positives divided by the number of True Positives and False Positives. Put another way, it is the number of positive predictions divided by the total number of positive class values predicted. It is also called the Positive Predictive Value (PPV).

- **Recall**: Recall is the number of True Positives divided by the number of True Positives and the number of False Negatives. Put another way it is the number of positive predictions divided by the number of positive class values in the test data. It is also called Sensitivity or the True Positive Rate. Recall can be thought of as a measure of a classifiers completeness. A low recall indicates many False Negatives.

- **Area under curve**: The AUC is the area under the ROC curve. This score gives us a good idea of how well the model performances. ROC (*Receiver Operating Characteristic*) Curve tells us about how good the model can distinguish between two things (*e.g If an application is a malware or not*). Better models can accurately distinguish between the two. Whereas, a poor model will have difficulties in distinguishing between the two.

- **F1 score**: The F1 Score is the 2*((precision*recall)/(precision+recall)). It is also called the F Score or the F Measure. Put another way, the F1 score conveys the balance between the precision and the recall.

II. Analysis

Data Pre-processing and Extraction

As said earlier, our downloaded original dataset contains 5,560 applications from 179 different malware families. I also had a total of about 129,000 text files that contained descriptions about the application.

For each application, the Drebin dataset contains a text file. The text file describes all the properties of the application. Each property belongs to one of 8 categories (S1 to S8). Then a feature vector for this application can be the occurrence of each category in the text file. For example, one application can have a feature vector of 1-12-0-3-5-67-9-4. It means that S1 category occurs 1 time, S2 occurs 12 times and so on. Some feature vector like **activity, service_receiver, provider, and service** were grouped as one since they have a high level of correlation. This step was done with this **code** found on github.

After a count of our feature vector was done for each text file. I classified each text file as a malware/non-malware by writing a script to verify if the text file was found in our 179 different malware families which was downloaded with the data. This script was done together with two other coursemates(Sayo Makinwa and Garba Mariam).

Fig. 2 shows a view of what our final dataset looks like. As seen, we've got a lot of FALSE values(not malware) versus TRUE(malware) in our dataset. Fig. 1 is exactly a plot of our entire dataset. The columns are the number of times each appeared in the text file.

Figure 2

	Sha256	Feature	Permission	Activity	Intent	Api_call	Real_permission	Call	Url	Class
0	118cfa0ae355dba8b5297f897dc3124f5bea3ed58c1c0e168d7bd43bf26d404d	3	6	18	3	8	7	10	54	FALSE
1	e27e833dcd4ab095c5c9cf3ac2abdfe944fe9a0e5723b36713c857de125f8585	2	2	7	2	1	1	3	4	TRUE
2	48c9ba6a1746d69dd1afeff7f7e57e1182c3b10155e1dd1ba8b8220bb30f7102	3	2	21	2	3	2	2	11	FALSE
3	42a181498c569212e8724c273fb40af1dcead28f11fec38d7fea13d8b123dfd3	6	7	2	2	7	6	4	25	FALSE
4	fa72e152139d38f166108cc2496c61bc323969763ff117db20a2b305e0d46d17	10	15	13	3	10	9	9	9	TRUE
5	a6c5a378ac6030f1186b75b8d50f235b6efdf63cc873ce54820f5e196e22c786	1	5	6	3	9	8	10	13	FALSE
6	8445541753809e467427a5678b8b3d605b456cab64cab4df3a1c76ab610e7c19	2	4	2	2	7	6	7	28	FALSE
7	7948afd546144c7095522091c59a52a73b974b4872102d68585dec4fe25a6305	2	7	4	2	8	7	8	8	FALSE
8	bcfda0f5fdbd422aa306fcd26d873295cb5ce23d04bee99545be5ffad9c956a3	2	4	9	2	11	10	8	29	FALSE

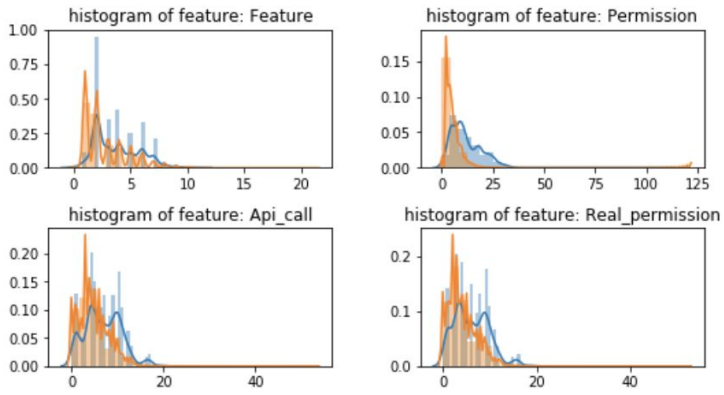
Data Exploration

In this study, we can conclude from Fig. 1 that the data is highly imbalanced. This dataset presents applications that have 5506 malwares out of 129,013. The dataset is highly unbalanced, the positive class (malware) account for about 0.18% of all the applications. It can also be concluded that all of the features extracted are discrete data.

Exploratory Visualization

An histogram was plotted for each feature and it was observed majority of the features were positively skewed, none had a normal distribution(Gaussian distribution). Fig. 3 shows four of the feature(**feature, permission, api_call and real_permission**) histogram.

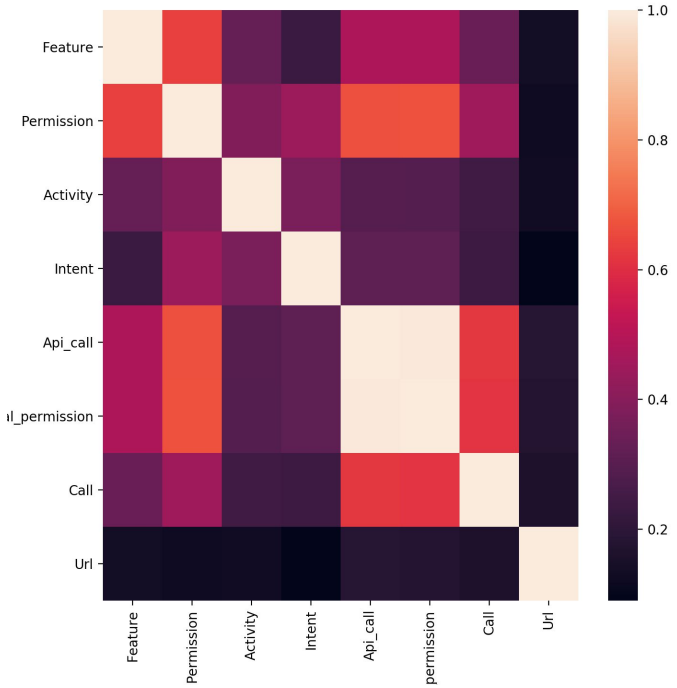
Figure 3



It can be seen the feature **api_call and real_permission which** have similar distributions between the two types of applications. It's important to note they were not the only pair with similar distributions. This discovery would enable me eliminate features that are equally skewed but since we had very few features(a total of 8), I decided not to eliminate any of the feature.

A feature correlation plot(Fig. 4) was done to view features correlation, this exploration again would have been useful to cut down features that were highly positively correlated but as seen, we have very few positively correlated features.

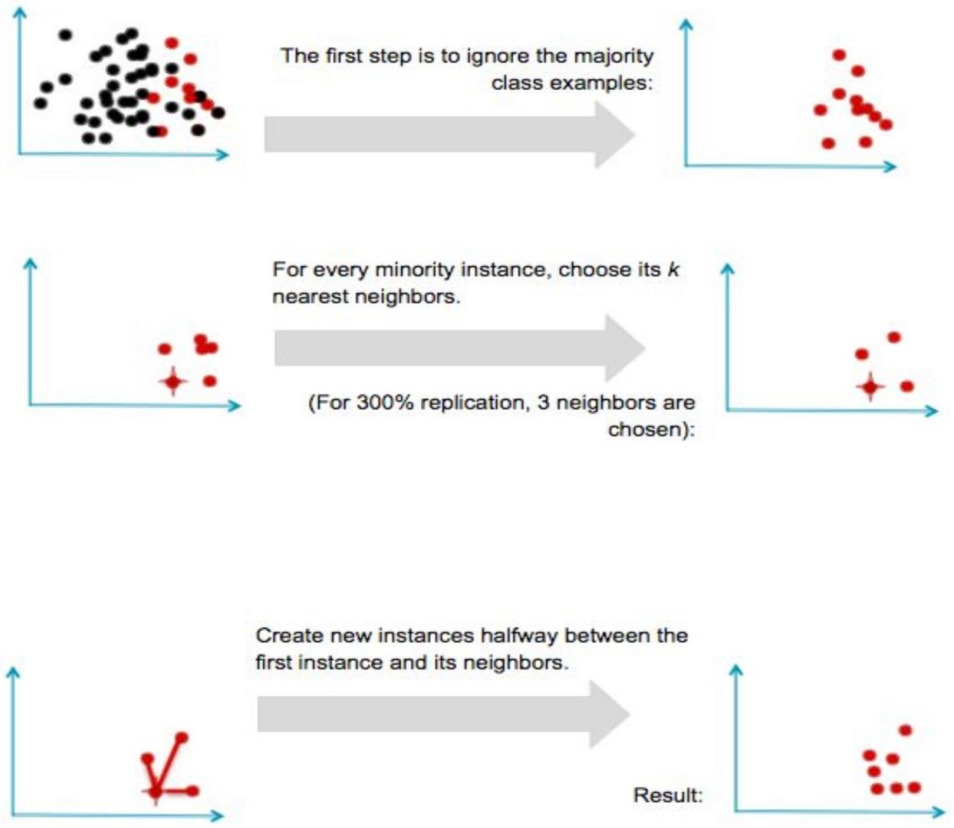
Figure 4



Algorithms and Techniques

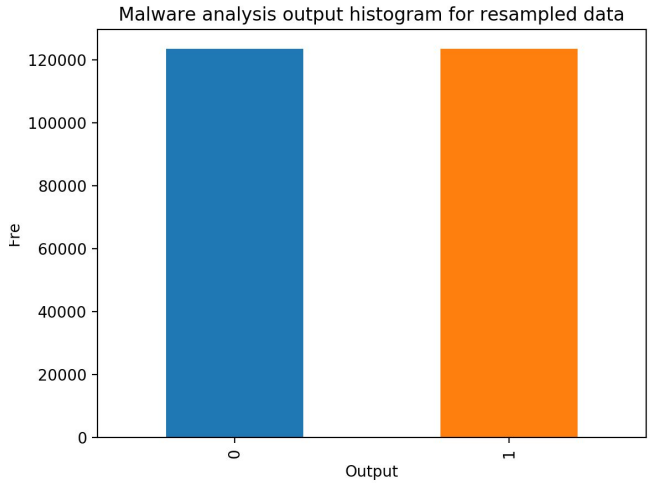
To solve our interesting imbalanced datasets problems, the SMOTE method was used since we couldn't get more data. This is a research method used in the synthesis of new datasets for imbalanced data like ours. The best known example of this approach is Chawla's SMOTE (Synthetic Minority Oversampling TEchnique) system. The idea is to create new minority examples by interpolating between existing ones. The process is well explained with Fig. 5 below.

Figure 5



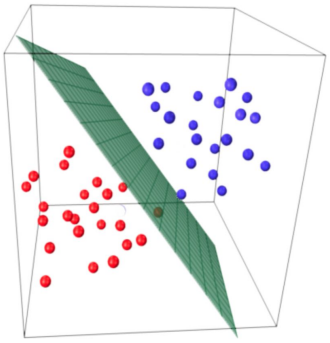
After applying the SMOTE method to produce new datasets, the total number of applications increased from 5506 to 246906 which is a significant increase. This gave a balance figure of 246906 for both the true and false class as seen below in comparison to Fig. 1

Figure 6



After a resampling of our datasets, I needed to focus on performance of the model and so ensemble classifiers and Logistic regression were adopted. I made use of the following algorithms: Logistic regression, RandomForestClassifier, and AdaBoostClassifier. Here are reasons for each algorithm selected:

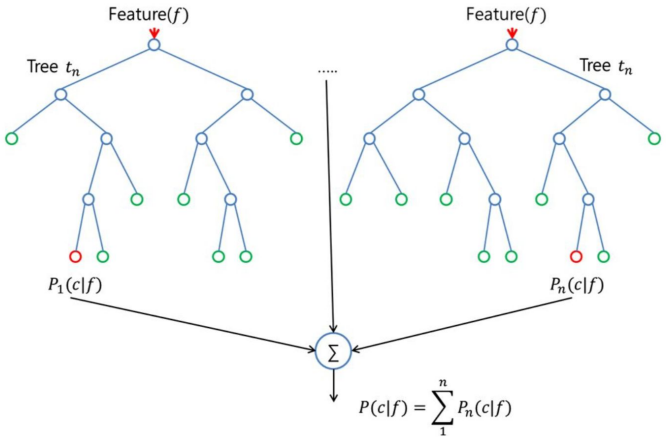
- **Logistic regression:** Logistic Regression is a type of classification algorithm involving a linear discriminant. Unlike actual regression, logistic regression does not try to predict the value of a numeric variable given a set of inputs. Instead, the output is a probability that the given input point belongs to a certain class, I selected this algorithm because we are dealing with a binary classification problem. For simplicity, let's assume that we



have only two classes and the probability in question is the probability that a certain data point belongs to the class. This dividing plane is called a ***linear discriminant***, because 1. its linear in terms of its function, and 2. it helps the model 'discriminate' between points belonging to different classes. Logistic regression has been said according to research that it doesn't perform well on imbalanced datasets but we will use it for a trial and error sake.

- **RandomForestClassifier:** According to this paper and other research journals, it is said that this classifier works perfectly on our type of datasets especially with the fact that it's training time is faster in terms of performance optimization.

It builds multiple such decision tree and amalgamate them together to get a more accurate and stable prediction. This is direct consequence of the fact that by maximum voting from a panel of independent judges, we get the final prediction better than the best judge. The implementation is illustrated in the figure above.



- **AdaBoostClassifier:** AdaBoost is a type of "Ensemble Learning" where multiple learners are employed to build a stronger learning algorithm. AdaBoost works by choosing a base algorithm (e.g. decision trees) and iteratively improving it by accounting for the incorrectly

> classified examples in the training set. We assign equal weights to all the training examples and choose a base algorithm. At each step of iteration, we apply the base algorithm to the training set and increase the weights of the incorrectly classified examples. We iterate n times, each time applying base learner on the training set with updated weights. The final model is the weighted sum of the n learners.

After examining each model training time without fine tuning any of the parameters, RandomForestClassifier and AdaBoostClassifier were further selected to be tested since they have the lowest training time and higher scores in the metrics. It's no surprise Logistic regression had the last position since it's been said not to be good for imbalanced data.

Benchmark

To benchmark the models, I used both RandomForestClassifier and AdaBoostClassifier to make predictions on the initial test datasets(after it was resampled with SMOTE). RandomForestClassifier and AdaBoostClassifier **precision_score** was 0.98 and 0.86 respectively.

III. Methodology

Data preprocessing

I also applied cross validation to split the datasets into training and test sets. 67% for training while the rest of the 33% goes for testing. I used the **stratify train_test_split**, this cross-validation method is a quick utility that wraps input validation with equal number of output label across each split.

Implementation

RandomForestClassifier was selected as the final model with the shortest training time, highest precision, recall, auc and f1_score of **0.983, 0.990, 0.986 and 0.986**. This classifier is an ensemble tool which takes a subset of observations and a subset of variables to build several decision trees.

Random forest is generally as a black box which takes in input and gives out predictions, without worrying too much about what calculations are going on the back end. This black box itself have a few levers we can play with. Each of these levers have some effect on either the performance of the model or the resource – time balance.

IV. Results

Model evaluation and validation

During development, a validation set was used to evaluate the model. I split the data into training and test sets. To verify the robustness of the model, I use it on the whole datasets to ensure it

performs great on both the train and test sets. The RFC model consistently categorized the applications as either malware or non-malware with an average precision score of **89%**.

Justification

This problem is highly imbalanced classification problem, where we have more than 99% transactions as malware and <1% as non-malware. Given data skewness, classifiers will tend to prefer normal (negative class) transactions and will have a challenge identifying malware positive classes. Hence to overcome this, SMOTE (Synthetic Minority Over -sampling Technique) method was used to oversample training data with positive classes, so that classifiers have more positive class to work with.

Ensemble methods (Adaboost, RandomForest) and Logistic Regression were trained on data as ensemble methods tend to predict better for these kinds of problems. Despite the fact that the data is highly imbalanced, RandomForest was able to achieve an average_precision_score of **0.98** while training on 74072 which is a fair score.

V. Conclusion

The most interesting part of this project for me is using the SMOTE technique to generates more data and also the use of various classifiers and how they perform before settling for the best classifier.

Reflection

The process for this project include the following:

1. Download and pre-process the data.
2. Identify the best metrics to benchmark our classifier.
3. Solve the imbalanced data problem.
4. Research various algorithms and techniques to solve the problem.
5. Implement a classifier and train it using the data.

Improvement

I believe there are several ways of improvement:

1. One way is to explore undersample; and also the combination of **undersampling** and **oversampling**.
2. Apply statistical techniques such as **square root or log transformation** to transform the **positively skewed distribution** as seen in Fig.3 to a no-skew(normal distribution).
3. **Hyperparameter tuning** and the application of **grid search** across the selected models in order to select the best parameter values will be an improvement to this project.
4. I can also do more work on data exploratory by plotting a **feature importance** graph to compare how each feature is relevant to each model.

References

- https://svds.com/learning-imbalanced-classes
- http://contrib.scikit-learn.org/imbalanced-learn/generated/imblearn.over_sampling.SMOTE.html
- https://www.analyticsvidhya.com/blog/2015/06/tuning-random-forest-model/
- http://machinelearningmastery.com/classification-accuracy-is-not-enough-more-performance-measures-you-can-use/
- https://en.wikipedia.org/wiki/Malware_analysis